Endbericht der PG 462: Optimierungsalgorithmen für Simulationsmodelle (OaSe)

PG 462

WS 2004/2005 und SS 2005

Veranstaltet vom Lehrstuhl für praktische Informatik (Modellierung und Simulation).

PG 462 sind:
Katharina Balzer, Marcus van Elst, Bartosz Fabianowski, Igor Gudovsikov,
Julia Hielscher, Christian Horoba, Thomas Hutter, Peter Kissmann, Jan Sören Kriege,
Normann Powierski, Michael Schaten,
(nur im ersten PG-Semester: Christian Bäcker)

Betreuer:
Dr. Peter Kemper, Dr. Axel Thümmler, Carsten Tepper

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1. Einleitung

Normann Powierski

1.1. Motivation

Die Aufgabenstellung dieser Projektgruppe umfaßt die Realisierung von Algorithmen zur Optimierung von Simulationsmodellen für diskrete ereignisorientierte Systeme. Bei der Optimierung wird eine Belegung der Eingabeparameter des Simulationsmodells gesucht, welche eine oder mehrere Ausgabegrößen des Modells minimiert bzw. maximiert.

Ein wesentliches Merkmal von Optimierungsalgorithmen für Simulationsmodelle ist, daß diese mit den mehr oder weniger starken stochastischen Schwankungen der Simulationsausgabe zu Recht kommen müssen. Ferner ist oft schon die Auswertung einer Parameterbelegung durch die Simulation sehr zeitaufwendig, so daß Optimierungsverfahren, die mit möglichst wenigen Auswertungen auskommen, von Interesse sind.

Auch wenn für die Optimierung von Simulationsmodellen zahlreiche Techniken existieren, treten bei der praktischen Anwendung doch zahlreiche Probleme auf, da viele der vorgeschlagenen Optimierungsansätze nicht robust genug sind, nicht für große Systeme oder zahlreiche Eingabeparameter skalieren, spezielle Anforderungen an Simulationsmodelle stellen, die in der Regel nicht erfüllbar sind, oder aber manuelle Eingriffe des Modellierers erfordern.

Die automatische Kopplung von ereignisdiskreten Simulatoren mit klassischen Optimierern führt aber gerade bei komplexen Simulationsmodellen meist zu unbefriedigenden Ergebnissen, wenn sie sich auf ein einfaches Zusammenführen von Simulationsmodell und Optimierungsverfahren beschränkt. Als robuste Methode, die das Potential einer solchen Kopplung mitbringt, bietet sich die seit langem bekannte *Response Surface Methode* und die Verwendung von *Evolutionsstrategien* oder *genetischen Algorithmen* an.

1.2. Gliederung des Endberichtes

Das Ziel der Projektgruppe OaSe ist die Realisierung von Algorithmen zur Optimierung von Simulationsmodellen. Um den Einstieg in das Thema zu erleichtern werden zunächst einige Grundlagen vorgestellt. Im weiteren gliedert sich der Bericht in die folgenden drei Teile:

- Optimierung
- Simulation
- Aufbau des Programms

Im Bereich Optimierung wird zunächst auf die von der Projektgruppe behandelten Optimierungsverfahren eingegangen. Die Methoden werden sowohl theoretisch vorgestellt als auch ihre Umsetzung und Performance beschrieben. Im einzelnen handelt es sich dabei um die Response Surface Methode, Evolutionäre Algorithmen, Memetische Algorithmen, Ranking & Selection Verfahren und Kriging Metamodelle.
Der Abschnitt Simulation stellt die verwendeten Simulationswerkzeuge ProC/B Toolset und APNN-Toolbox vor. Anschließend werden die bearbeiteten Simulationsmodelle erläutert und die Ergebnisse ihrer Optimierung mit Hilfe der vorher beschriebenen Verfahren gezeigt.
Zu guter Letzt wird im dritten Abschnitt erklärt, wie die verschiedenen Komponenten für Optimierung und Simulation im Programm „OaSe“ zusammenspielen.

2. Grundlagen

2.1. Optimierung von Simulationsmodellen

Normann Powierski, (Christian Bäcker)

Dieser Abschnitt stellt einen kurzen Überblick über die Grundlagen und die Probleme bei der Simulationsoptimierung dar.

2.1.1. Grundlagen

2.1.1.1. Der Systembegriff

Unter einem System versteht man eine Anzahl in Beziehung stehender Komponenten, die zu einem gemeinsamen Zweck interagieren. Die einzelnen Komponenten können dabei wiederrum aus Unterkomponenten zusammengesetzt sein. Komponenten werden durch ihre Eigenschaften definiert. Der Zustand einer Komponente oder eines Systems wiederum wird durch veränderbare Eigenschaften, sogenannte Zustandsvariablen, bestimmt. Man unterscheidet zwischen natürlichen und künstlichen, offenen und geschlossenen sowie statischen und dynamischen Systemen.

2.1.1.2. Der Modellbegriff

Ein gegebenes System kann oftmals nicht analytisch erfaßt werden, da seine Faktoren viel zu komplex, oder aber teilweise unbekannt sind. Aus diesem Grund erschafft man Ersatzsysteme, sogenannte Modelle, die das System ausreichend genau abbilden, aber trotzdem handhabbar sind. Man unterscheidet zwischen statischen und dynamischen, deterministischen und stochstischen sowie kontinuierlichen und diskreten Modellen.

2.1.1.3. Simulation

Unter dem Begriff Simulation versteht man die Durchführung von Experimenten an einem Modell, um entweder das Systemverhalten zu verstehen oder verschiedene Strategien für Systemoperationen zu gewinnen. Es wird dabei das Verhalten des realen Systems imitiert.
Bei der Simulation sind also Eingaben und Modell bekannt. Bestimmt werden müssen somit die Ausgaben des Modells.

2.1.1.4. Optimierung

Bei der Optimierung versucht man eine Menge von gültigen Eingaben zu finden, für die die Ausgaben des Modells optimal sind. In der Regel versucht man bestimmte Ausgaben

zu minimieren (bzw. maximieren).

2.1.2. Was macht die Optimierung von Simulationsmodellen so schwierig?

Gegenüber gewöhnlichen deterministischen Optimierungsalgorithmen, machen die stochastischen Schwankungen bei der Optimierung von Simulationsmodellen die größten Schwierigkeiten.
Realistische Simulationsmodelle können wegen einer Vielzahl von verschiedenen Variablen sehr komplex werden. Wenn die Struktur der objektiven Funktion dann noch unbekannt ist, wird die Optimierung durch Simulation noch schwieriger. Dies liegt daran, daß die Performance (Qualität) eines Entwurfs nicht exakt berechnet werden kann, sondern geschätzt werden muß. Wegen den Schätzungen kann man dann auch nicht genau sagen, ob ein Entwurf tatsächlich besser als ein anderer ist.
Prinzipiell läßt sich dieses Problem umgehen, wenn für wenige Entwürfe so viele Simulationsläufe gemacht würden, daß die Performance-Schätzungen nicht mehr variieren. Praktisch würde dies für komplizierte Probleme allerdings bedeuten, daß nur einige wenige verschiedene Entwürfe berechnet werden können, da ein Simulationsdurchlauf entsprechend viel Rechenzeit verbraucht, und eine genauere Betrachtung jegliche Zeitrahmen sprengen würde.

2.1.3. Simulationsergebnisanalyse für ein einzelnes System

2.1.3.1. Übergangs- und statisches Verhalten stochastischer Prozesse

Betrachten wir den zeitlichen Ablauf von Simulationen, so werden wir feststellen, daß die Verteilungen der Zufallsvariablen zu Beginn noch von den Anfangsparametern abhängen bis die Simulation einen (quasi) statischen Zustand erreicht hat.

Hat eine Simulation nach einer bestimmten Zeit den statischen Zustand erreicht werden die Verteilungen der Zufallsvariablen der verschiedenen Meßreihen unabhängig von ihren Startparametern annähernd die gleiche Verteilung haben. Dies bedeutet, die Werte die die Zufallsvariablen annehmen, werden nicht mehr von den Startparametern beeinflußt. Dies bedeutet jedoch nicht, daß die Zufallsvariablen der einzelnen Versuchsreihen die gleichen Werte annehmen müssen.
Abbildung 2.1 verdeutlicht das Übergangsverhalten von Simulationen.

2.1.3.2. Verschiedene Typen von Simulationen

Abhängig von der Art der Daten, die aus einer Simulation gewonnen werden sollen und abhängig von dem zu simulierenden Prozeß, hat man verschiedene Möglichkeiten für einen Entwurf.
Endende Simulationen werden durch ein bestimmtes Ereignis E abgeschlossen. Dies kann z.B. das verstreichen einer bestimmten Zeitperiode sein oder auch das Erreichen eines bestimmten Zustands. Relevant sind die Daten nach Abschluß der Simulation.
Nicht endende Simulationen werden durch kein Ereignis beendet, hier muß man den Zeitpunkt der Datengewinnung daher anders wählen. Simulationen die einen statischen

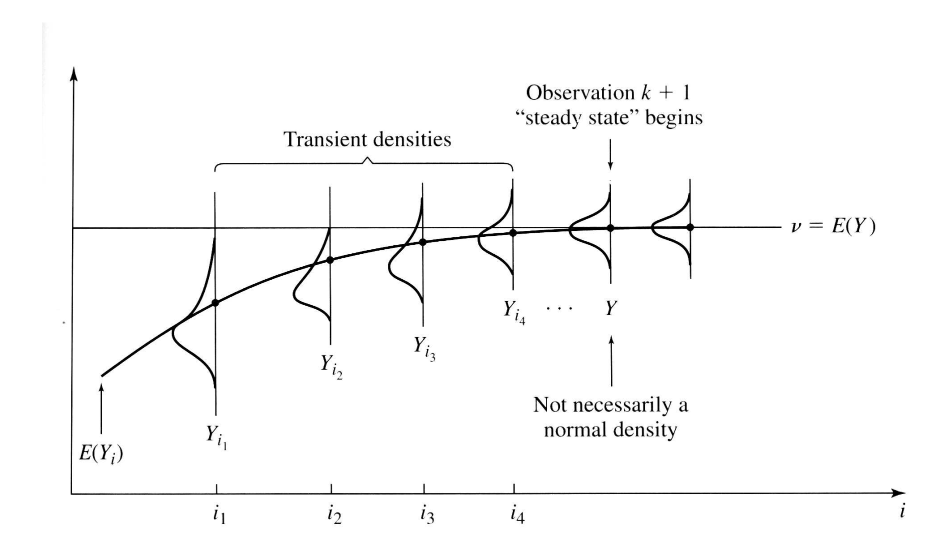


Abb. 2.1.: Übergangsverhalten von Simulationen

Zustand erreichen eignen sich z.B. um Leistungsdaten im normalen Betrieb eines Systems zu ermitteln.
Bei **Zyklischen Simulationen**, d.h. bei Simulationen, die nicht enden, bei denen aber bestimmte Konfigurationen zu immer gleichen Zeitpunkten wieder auftreten, lassen sich Daten jeweils nach beenden eines Zyklus gewinnen.

Ändern sich die Parameter einer Simulation fortlaufend, so daß diese weder einen statischen Zustand erreicht noch in regelmäßigen Abständen den gleichen Zustand erreicht, gibt es eigentlich keinen sinnvollen Zeitpunkt während der Simulation um Daten zu gewinnen. Man wählt hier den Zeitpunkt der Datenerfassung so, daß die Menge der gelieferten Daten ausreicht um die die Änderungen der Parameter über die Zeit zu bestimmen.

2.2. Mathematische Grundlagen für die Analyse von Simulationsmodellen

Normann Powierski

2.2.1. Einleitung

Obwohl Simulationen ein praktisches und experimentelles Verfahren sind, kann man ohne die Kenntnis mathematischer Methoden und Hintergründe, keine zuverlässigen Rückschlüsse aus Simulationsergebnissen ziehen. Auch für die Konstruktion probabilistischer Systeme ist fundiertes Wissen über stochastischer Verfahren unumgänglich. Im Rahmen dieses Abschnittes werden einige Grundlagen zur Auswertung von zufallsbeeinflußten Werten wie Simulationsergebnissen vermittelt.

2.2.2. Diskrete und kontinuierliche Zufallsvariablen

Ein Experiment ist ein Vorgang dessen Ausgang wir nicht mit Sicherheit vorhersagen können. Die Menge aller möglichen Ausgänge eines Experiments, nennt man Ereignisraum (S), die einzelnen Ausgänge selbst (Elementar-)Ereignisse.
Eine Zufallsvariable X ist eine Funktion, die jedem (Elementar-)ereignis eines Experimentes eine reelle Zahl zuweißt. Jedes zufällige Ereignis eines Versuches wird also durch X beschrieben. Zufallsvariablen werden für gewöhnlich mit $X, Y, Z, \ldots$ benannt, die Werte die sie annehmen mit den entsprechenden Kleinbuchstaben $x, y, z, \ldots$. Ist die Menge der Werte die eine ZV annehmen kann endlich bzw. abzählbar, spricht man von einer diskreten Zufallsgröße. Ist die Menge überabzählbar (z.B. $\mathbb{R}^{<0}$) von einer kontinuierlichen Zufallsgröße.

2.2.2.1. Darstellung von Zufallsvariablen

Diskrete Zufallsvariablen werden durch Angabe der Wahrscheinlichkeiten für ihre möglichen Ausgänge dargestellt. Nicht angegebene Wahrscheinlichkeiten sind gleich 0. Kontinuierliche Zufallsvariablen werden über die Wahrscheinlichkeitsdichte definiert; aus dieser können alle Wahrscheinlichkeitswerte einer ZV berechnet werden. Die Wahrscheinlichkeit dafür, daß der Wert einer ZV in einem bestimmten Bereich liegt, entspricht dabei der Fläche zwischen Kurve und x-Achse in diesem Bereich. Die Wahrscheinlichkeit für einen einzelnen Wert ist somit gleich 0.

Die (kumulative) Verteilungsfunktion $F(x)$ einer Zufallsvariable gibt an wie hoch die Wahrscheinlichkeit dafür ist, daß ihr Wert kleiner oder gleich x ist. Also

$$F(x) = P(X \leq x) \text{ für } -\infty < x < \infty$$

Bei diskreten Zufallsvariablen:

$$F(x) = \sum_{x_i \leq x}^{p(x_i)}$$

Bei kontinuierlichen Zufallsvariablen:

$$F(x) = \int_{-\infty}^{x} f(y)\, dy$$

wobei $f(y)$ die *Wahrscheinlichkeitsdichte* ist.

2.2.2.2. Mittelwert und Median

Der Mittelwert oder Erwartungswert ist ein Maß für die Tendenz einer Zufallsvariable. Macht man unendlich viele Versuche, wird deren mittleres Ergebnis sich dem Mittelwert angleichen.
Für diskrete Zufallsvariablen gilt:

$$\mu = \sum_{j=1}^{\infty} x_j p_x(x_j)$$

Für kontinuierliche Zufallsvariablen entsprechend:

$$\mu = \int_{-\infty}^{\infty} xf(x)\ dx$$

Ein Nachteil dieses Mittelwertes ist, daß er sehr stark von großen Zahlen beeinflusst wird auch, wenn diese nur mit sehr geringer Wahrscheinlichkeit auftreten. Daher gibt es ein anderes Maß für die Tendenz einer ZV, für das dieses Problem nicht zutrifft, den Median. Der Median einer ZV ist definiert als der kleinste Wert, so daß die Wahrscheinlichkeit dafür, daß die ZV einen Wert kleiner oder gleich diesem Wert annimmt, gleich $0,5$ ist. Somit fällt der Median im Gegensatz zum Mittelwert immer mit einem konkreten Wert der ZV zusammen.

2.2.2.3. Varianz und Standardabweichung

Kennt man den Mittelwert einer Zufallsvariable, so kann man eine Aussage über deren zentrale Tendenz treffen. Eine weitere Charakteristik einer ZV ist deren Varianz σ^2. Diese ist ein Maß dafür wie wahrscheinlich es ist, daß der Wert einer ZV dicht bei Ihrem Mittelwert liegt. Die Varianz $Var(X)$ oder σ^2 ist definiert als

$$\sigma^2 = E\left(X^2\right) - E(X)^2$$

2.2.3. Mehrere Zufallsvariablen

Für die meisten Simulationsmodelle reicht die Betrachtung einer einzelnen Zufallsvariable nicht aus. Bei Betrachtung mehrerer Zufallsvariablen in einem Modell ist es von besonderer Bedeutung, ob der Wert einer Zufallsvariablen von dem einer anderen abhängt oder ob es sich um unabhängige Zufallsvariablen handelt. Weiterhin ist es von Bedeutung, ob die ZVs in die gleiche (bzw. genau entgegen gesetzte) Richtung tendieren, d.h. in Korrelation zueinander stehen oder völlig unkorreliert sind.

2.2.3.1. Abhängigkeit zweier Zufallsvariablen

Zwei Zufallsvariablen (X, Y) mit gemeinsamer Zufallsfunktion
$p(x,\ y) =\ P(X =\ x,\ Y =\ y)$ sind unabhängig, wenn

$$p_x(x) = \sum_{\forall y} p(x,y) \quad \text{und} \quad p_y(y) = \sum_{\forall x} p(x,y)$$

Dies bedeutet, daß die Wahrscheinlichkeit, daß $X = x$ eintritt, wenn man y außer acht läßt, genauso groß sein muß, wie die Summe der Wahrscheinlichkeiten aller möglichen Kombinationen von $X = x$ und $Y = beliebig$. Analog sind 2 kontinuierliche Zufallsvariablen mit gemeinsamer Wahrscheinlichkeitsdichtefunktion $P(X \in A, Y \in B) = \int_B \int_A f(x,y)dxdy$ unabhängig, wenn

$$f(x,y) = f_x(x)f_y(y) \quad \forall x, y$$

wobei

$$f_x(x) = \int_{-\infty}^{\infty} f(x,y)dy \quad \text{und} \quad f_y(y) = \int_{-\infty}^{\infty} f(x,y)dx$$

Anschaulich, ist die Wahrscheinlichkeit, daß X in einem bestimmten Bereich A und Y in B liegt, gleich dem Volumen unterhalb der Funktion f in diesem Bereich. Somit ist $f_x(X = A)$ entsprechend das Volumen für $X \in A$ für Y im Intervall $[-\infty, \infty]$.

2.2.3.2. Kovarianz und Korrelation

Die Kovarianz ist ein Maß für die (lineare) Abhängigkeit zweier Variablen. Sie ist definiert als

$$C_{ij} = E\left[(X_i - \mu_i)(X_j - \mu_j)\right] = E(X_i X_j) - \mu_i \mu_j$$

Bei positiver Kovarianz tendieren beide Zufallsvariablen in die gleiche Richtung (beide oberhalb oder beide unterhalb ihres Mittelwertes). Bei negativer Kovarianz tendieren sie in entgegen gesetzte Richtungen. Ist die Kovarianz gleich 0 gibt es keine lineare Abhängigkeit zwischen den Zufallsvariablen.

Während der Wert der Kovarianz keinen direkten Rückschluß auf den Grad der Abhängigkeit zuläßt, da er von der Dimension der Zufallsvariablen abhängt, kann man am Wert der Korrelation den Grad der Abhängigkeit direkt erkennen.

$$Cor(X,Y) = \frac{Cov(X,Y)}{\sigma_X \sigma_Y}$$

Die Korrelation (ρ) zweier Variablen bewegt sich immer zwischen -1 und $+1$. Bei Werten nahe 1 spricht man von stark positiv korrelierten Variablen, nahe -1, von stark negativ korreliert und nahe 0 von schwach, bzw. unkorrelierten Variablen. (Vergleiche Abbildung 2.2)

2.2.4. Auswertung von Simulationsergebnissen

Während wir bisher Zufallsvariablen betrachtet haben, deren Verteilung bekannt ist, so daß ihre Charakteristika wie Mittelwert und Varianz direkt berechnet werden können, betrachten wir jetzt Zufallsvariablen, bei denen wir nur über einige (konkrete) Werte aus dem Ereignisraum verfügen. Derartige Daten können z.B. Ergebnisse einer Simulation sein, aber auch Meßwerte aus einem Versuch.

2.2.4.1. Stochastische Prozesse

Ein stochastischer Prozeß ist eine Menge von Zufallsvariablen mit derselben Wertemenge, dem Zustandsraum. Wir unterscheiden zwischen stochastischen Prozessen mit diskreter Zeit (Zufallsvariablen $X_1, X_2, \ldots$) und stochastischen Prozessen mit kontinuierlicher bzw. stetiger Zeit $(X(t), t \geq 0)$.
Man spricht von *kovarianzstationären* Prozessen, wenn der Mittelwert und die Varianz

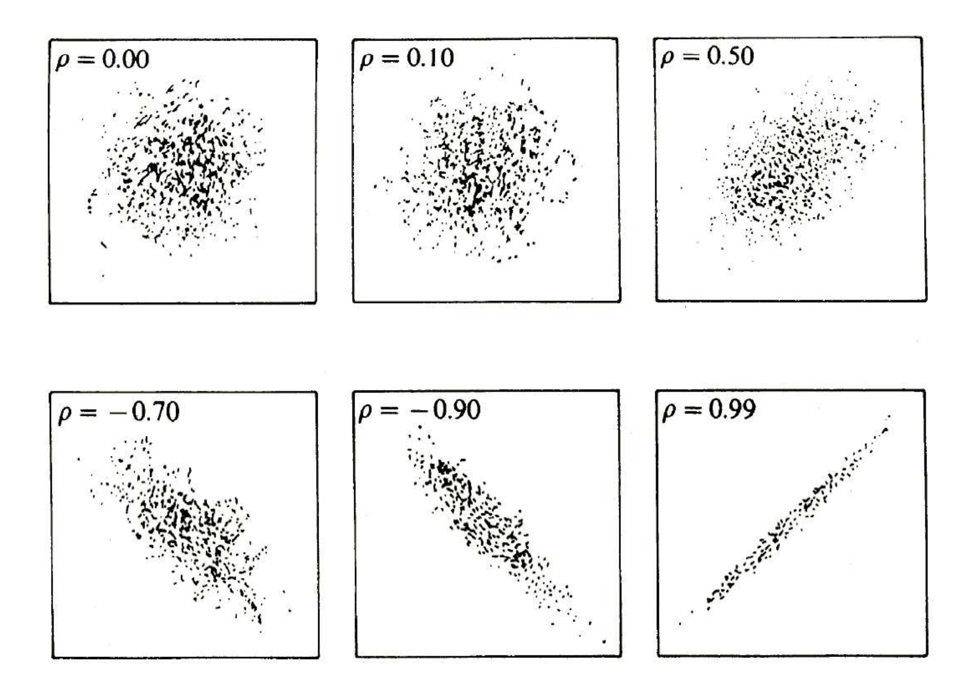


Abb. 2.2.: Korrelation zweier ZV. Jeder Punkt entspricht einer Messung von X und Y.

über die Zeit konstant bleiben und die Kovarianz zwischen zwei Variablen im selben Abstand immer gleich ist.

$$\mu_i = \mu \; ; \quad \sigma_i^2 = \sigma^2 \qquad \text{für} \quad i = 1, 2, \ldots$$
$$C_{i,i+j} = cov[X_i, X_{i+j}] \quad \text{für} \quad i = 1, 2, \ldots \quad j = 0, 1, 2, \ldots \quad \text{unabhängig von i}$$

Daher ist die Kovarianz hier gleich C_j.

2.2.4.2. Schätzer und Schätzfunktionen

Als Grundgesamtheit versteht man die Menge aller Elemente, die auf bestimmte Merkmale untersucht werden sollen. Z.B. alle Meßwerte einer Meßreihe, alle Stücke einer Produktionsreihe oder ähnliches. Da die Menge der Elemente der Grundgesamtheit sehr groß oder sogar unendlich groß sein kann, entnimmt man der Grundgesamtheit eine Teilmenge, eine so genannte Stichprobe. In der Regel sollte man darauf achten, daß jedes Element der Grundgesamtheit die gleiche Chance hat in die Stichprobe zu gelangen. Man spricht dann von einer zufälligen Stichprobe. Realisierbar ist dies z.B. durch Festlegung der Elemente durch Zufallszahlen.

2.2.4.3. Zufallsvektoren

Faßt man mehrere Zufallsvariablen zusammen, z.B. die Werte aus mehreren Stichprobenreihen, erhält man einen Zufallsvektor. $\vec{X} = (X_1, X_2, \ldots, X_n)$. Jede konkrete Stichprobe

aus einer Grundgesamtheit kann als Vektor geschrieben werden $\vec{x} = (x_1, x_2, \ldots, x_n)$.

2.2.4.4. Stichprobenfunktionen

Der Mittelwert eines Zufallsvektors lautet:

$$\overline{X}(n) = \frac{1}{n} \sum_{i=1}^{n} X_i$$

wobei n die Anzahl der Experimente ist, die jeweils ein X_i zum Ergebnis haben.

Da wir die Wahrscheinlichkeit für das Eintreten eines bestimmten Wertes nicht kennen, wird hier Gleichverteilung angenommen. Das heißt, die Wahrscheinlichkeit für jedes Element ist $1/n$.

Die (Stichproben-)Varianz berechnet sich entsprechend zu

$$S^2(n) = \frac{1}{n-1} \sum_{i=1}^{n} [X_i - \overline{X}(n)]^2$$

2.2.4.5. Vertrauensintervalle

2.2.4.5.1. Abschätzen der Varianz

Da es sich bei $\overline{X}(n)$ um eine Zufallsvariable mit Varianz $Var\left[\overline{X}(n)\right]$ handelt, wird der Mittelwert μ verschiedener Experimente mehr oder weniger stark von $\bar{X}(n)$ abweichen. Um eine Aussage über die Genauigkeit von $\bar{X}(n)$ treffen zu können, bestimmt man daher Vertrauens- bzw. Konfidenzintervalle.

Erster Schritt zur Bestimmung eines Vertrauensintervalls ist die Bestimmung der Varianz des Mittelwertes $\bar{X}(n)$. Denn auch bei sehr vielen Stichproben bleibt $\bar{X}(n)$ eine (kontinuierliche) Zufallsgröße.

$$\begin{aligned} Var\left[\overline{X}(n)\right] &= Var\left(\frac{1}{n}\sum_{i=1}^{n} X_i\right) \\ &\vdots \\ \frac{1}{n^2} n\sigma^2 &= \frac{\sigma^2}{n} \end{aligned}$$

Zieht man nur eine einzige Zahl, entspricht die Varianz des Mittelwertes grade der Varianz der Grundgesamtheit. Bei einer normalverteilten Grundgesamtheit, ergibt sich somit für den geschätzten Mittelwert die gleiche Normalverteilung. Mit steigender Anzahl von Werten in einer Stichprobe, sinkt die Streuung des Mittelwertes. Bei nicht normalverteilten Grundgesamtheiten ergibt sich für $\overline{X}(n)$ dennoch eine Normalverteilung.

Da die Varianz der Grundgesamtheit σ^2 in der Regel nicht bekannt ist, ersetzen wir diese durch die Stichprobenvarianz $S^2(n)$ und erhalten einen erwartungstreuen Schätzer

für die Varianz des Mittelwertes.

$$\hat{V}ar[\overline{X}(n)] = \frac{S^2(n)}{n} = \frac{\sum_{i=1}^{n}[X_i - \overline{X}(n)]^2}{n(n-1)}$$

2.2.4.5.2. Abschätzen der Varianz für abhängige Zufallsvariablen

Die bisher verwendeten X_i sind unabhängig und unkorreliert. Im Gegensatz dazu sind aus Simulationen gewonnene Daten erfahrungsgemäß nahezu immer korreliert.

Nehmen wir als Beispiel kovarianzstationäre Variablen. Bei diesen bleibt zwar der Schätzer des Mittelwerts $\overline{X}(n)$ erwartungstreu, nicht jedoch $S^2(n)$ als Schätzer für die Varianz der Grundgesamtheit.
Es kann gezeigt werden, daß

$$E[S^2(n)] = \sigma^2 \left[1 - 2\frac{\sum_{j=1}^{n-1}\left(1-\frac{j}{n}\right)\rho_j}{n-1} \right] \quad (\rho_j = \text{Korrelation zwischen 2 ZVs im Abstand } j)$$

Dies bedeutet, daß bei kovarianzstationären Variablen der Schätzer gegenüber dem ,,richtigen“ σ^2 mit steigendem n nach unten hin abweichen wird.

2.2.4.5.3. Konfidenzintervalle

Wie oben bereits erwähnt nähert sich die Verteilung von $\bar{X}(n)$ für große n einer Normalverteilung an. Sei $Z_n$die Zufallsvariable $\frac{\bar{X}(n)-\mu}{\sqrt{\sigma^2/n}}$ und F_n(z) ihre Verteilungsfunktion. Dann besagt der Grenzwertsatz, daß sich die Verteilung von Z_n mit steigendem n der Verteilung einer standardnormalverteilten Variablen (siehe Abbildung 2.3) annähert.

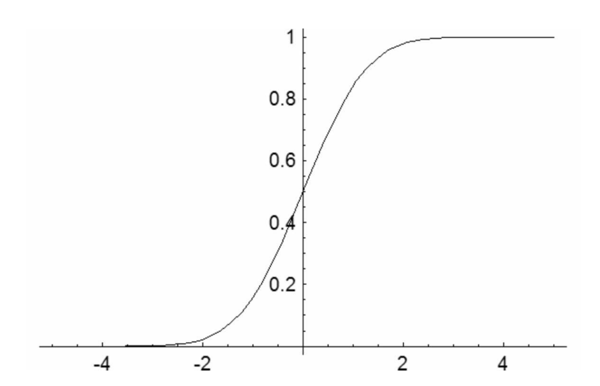


Abb. 2.3.: Verteilung einer standardnormal verteilten Variablen. Mittelwert $\mu = 0$ und Varianz $\sigma^2 = 1$

$\bar{X}(n)$ selbst nähert sich einer Normalverteilung mit Mittelwert μ und Varianz σ^2/n an (siehe Abbildung 2.4).

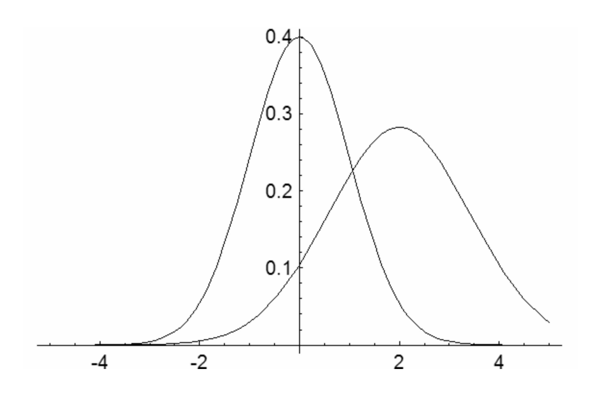


Abb. 2.4.: Dichtefunktion für Standardnormalverteilung und $\overline{X}$(n)

Dadurch, daß Z_n für große n annähernd eine standardnormalverteilte Variable ist, kann man die Wahrscheinlichkeit dafür, daß Z_n in einem bestimmten Bereich liegt bestimmen:

$$P\left(-z_{1-\alpha/2} \leq Z_n \leq z_{1-\alpha/2}\right) \Leftrightarrow P\left(-z_{1-\alpha/2} \leq \frac{\bar{X}(n)-\mu}{\sqrt{\sigma^2/n}} \leq z_{1-\alpha/2}\right) \approx 1-\alpha$$

Da die Stichprobenvarianz erwartungstreuer Schätzer für σ^2 ist können wir σ^2 für große n gegen S^2(n) ersetzen. Dies ist notwendig, da σ^2 im Allgemeinen nicht bekannt ist.

$$P\left(-z_{1-\alpha/2} \leq \frac{\bar{X}(n)-\mu}{\sqrt{S^2(n)/n}} \leq z_{1-\alpha/2}\right) \approx 1-\alpha$$

Formt man diesen Term noch etwas um, erhält man auch eine direkte Aussage über die Wahrscheinlichkeit dafür, daß μ in einem bestimmten Bereich liegt.

$$P\left(\bar{X}(n) - z_{1-\alpha/2}\sqrt{S^2(n)/n} \leq \mu \leq \bar{X}(n) + z_{1-\alpha/2}\sqrt{S^2(n)/n}\right) \approx 1-\alpha$$

Daraus läßt sich dann ein Vertrauensintervall mit Wahrscheinlichkeit $100(1-\alpha)$ für $\overline{X}(n)$ bestimmen.

$$\bar{X}(n) \pm z_{1-\alpha/2}\sqrt{\frac{S^2(n)}{n}}$$

Dieses Vertrauensintervall kann aber nur als Annäherung gesehen werden. Wird die Anzahl der Stichproben n zu klein gewählt, wird die Verteilung von Z_n nicht mehr ausreichend einer standardnormal Verteilung entsprechen. Daher wählen wir statt z_1 die Zufallsvariable $t_n = \left[\bar{X}(n) - \mu\right]/\sqrt{S^2(n)/n}$

$$\bar{X}(n) \pm t_{n-1,1-\alpha/2}\sqrt{\frac{S^2(n)}{n}}$$

Bei anders als normalverteilten Zufallsvariablen, weicht der tatsächliche Wert der im Vertrauensintervall liegenden Mittelwerte vom errechneten Wert ab. Erst bei einer großen Anzahl von Stichproben, nähert er sich dem errechneten Wert an. Von besonderer Bedeutung ist dabei die Symmetrie einer Verteilung. Je unsymmetrischer (schiefer) die Verteilung, desto größer die Abweichung.

2.2.4.5.4. Statistische Tests (für den Mittelwert)

Nun wollen wir den umgekehrten Weg gehen und für einen angenommene Mittelwert μ_o herausfinden ob er gleich dem echten Mittelwert μ ist.
Wir wissen, daß falls $\mu_o = \mu$ ist die Abweichung von μ_o zu $\overline{X}(n)$ wahrscheinlich nicht zu groß sein wird. Da $\overline{X}(n)$ erwartungstreuer Schätzer von μ ist. Nun wissen wir, daß falls $\mu_o = \mu \quad \mu_0$ auch im Vertrauensintervall liegen sollte. Also können wir mit

$$t_n = \frac{\left[\overline{X}(n) - \mu_o\right]}{\sqrt{\frac{S^2(n)}{n}}}$$

überprüfen ob μ_0 innerhalb der kritischen Grenzen der Verteilung liegt.

2.3. Gemeinsame Zufallszahlen (Common Random Numbers)

Christian Horoba

In diesem Abschnitt wird eine bekannte varianzreduzierende Technik für den Vergleich zweier Konfigurationen für Simulationsmodelle beschrieben (siehe z. B. [14]). Man beachte, daß es sich hierbei um eine Heuristik handelt, die jedoch bei typischen Simulationsmodellen das angestrebte Ziel erreicht.

Im Allgemeinen besteht die Aufgabe statistischer Auwahlverfahren darin, aus einer Population von Zufallsgrößen $X_1, \dots, X_k$ eine Zufallsgröße mit dem kleinsten Erwartungswert auszuwählen. Um zwei Zufallsgrößen X_i und X_j miteinander zu vergleichen, ist der Erwartungswert der Zufallsgröße $X_i - X_j$ von Interesse. Dieser kann folgendermaßen geschätzt werden:

$$\frac{1}{n} \cdot \sum_{l=1}^{n} X_{il} - X_{jl},$$

wobei X_{ij} die j-te Realisation der Zufallsgröße X_i bezeichnet. Für die Varianz der Zufallsgröße $X_i - X_j$ gilt:

$$Var\left(X_i - X_j\right) = Var\left(X_i\right) + Var\left(X_j\right) - Cov\left(X_i, X_j\right).$$

Man kann die genannte Formel so deuten, daß eine positive Korrelation der betrachteten Zufallsgrößen die Breite eines Konfidenzintervalls für den Erwartungswert verkleinert, d. h. das im Allgemeinen weniger kostenintensive Funktionsauswertungen nötig sind.

Unter der Verwendung gemeinsamer Zufallszahlen versteht man den geschickten Einsatz von Zufallsströmen, um den geschilderten Zustand herzustellen. Für die Umsetzung dieser Technik ist es notwendig für jeden Ereignistyp des Simulationsmodells und jede Auswertungsiteration unabhängige Zufallsstöme zur Verfügung zu stellen. Es werden insbesondere identische Zufallsströme für die selbe Auswertungsiteration verschiedener zu vergleichender Konfigurationen verwendet.

Im Folgenden wird beschrieben, wie die Technik auf die Simulationsmodelle *(s,S)-Lagerhaltungssystem* und *Tandem-Warteschlange* angewendet werden kann.

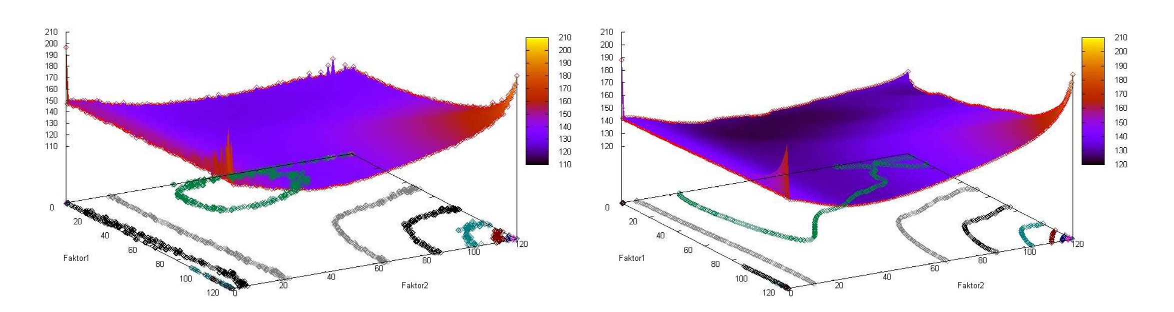


Abb. 2.5.: Oberfläche der Kostenfunktion des *(s,S)-Lagerhaltungssystems*: 1 Replikation (ohne CRNs) (l.), 1 Replikation (mit CRNs) (r.).

Für die Simulation des Modells *(s,S)-Lagerhaltungssystem* werden unabhängige Zufallsströme für die Bestimmung der folgenden Werte verwendet: Lieferzeiten der Bestellungen, Zwischenankunftszeiten der Kundenbestellungen, Bestellmengen der Kundenbestellungen. Für die Simulation der *Tandem-Warteschlange* werden die Zwischenankunftszeiten der Objekte am ersten Wartesystem und die Bedienzeiten der einzelnen Wartesysteme jeweils mit unabhängigen Zufallsströmen bestimmt.

Abbildung 2.5 stellt die Oberfläche des *(s,S)-Lagerhaltungssystems* (siehe Abschnitt 6.4.3.2) sowohl ohne als auch mit Einsatz von CRNs dar. Man beachte, daß die Oberfläche durch den Einsatz gemeinsamer Zufallszahlen geglättet wird.

Der Nutzen dieser Technik kann im entsprechenden Unterabschnitt der Leistungsstudien nachvollzogen werden.

2.4. Benchmarkfunktionen für Optimierungsalgorithmen

Bartosz Fabianowski

Bei einem Simulationsmodell ist die Auswertung für eine Faktorenbelegung sehr zeitaufwendig. Für die Entwicklung und Bewertung von Optimierungsverfahren ist diese lange Auswertungszeit problematisch, weil hierbei viele Optimierungsschritte mit vielen Auswertungen durchgeführt werden müssen. Es ist daher wünschenswert, statt Simulationsmodellen andere Modelle zu verwenden, deren Response schneller ausgewertet werden kann.

Besonders geeignet sind einfache mathematische Funktionen, die in geschlossener Form vorliegen und daher sehr effizient ausgewertet werden können. Wir haben für die Verwendung in unseren Leistungsstudien insgesamt neun solche Benchmarkfunktionen implementiert. Sie sind in Tabelle 2.1 zusammengefaßt. Die Spalten geben der Reihe nach den Namen, die Funktionsvorschrift und schließlich einen Beispielplot der Response Surface über $[-2, 2] \times [-2, 2]$ an.

Den Funktionen kann Rauschen unterschiedlicher Stärke und unterschiedlichen Typs zugeschaltet werden. Das Rauschen hat die Form einer normalverteilten Zufallsvariable.

Name	Funktionsvorschrift	Beispielplot
Sphere	$\sum_{i=1}^{n} x_i^2$	
Rosenbrock	$\sum_{i=1}^{n} \left(100\left(x_i^2 - x_{i+1}\right)^2 + (1-x_i)^2\right)$	
Step	$\sum_{i=1}^{n} \lfloor \lvert x_i \rvert + 0,5 \rfloor^2$	
Ellipsoid	$\sum_{i=1}^{n} i x_i^2$	
Schwefels Ridge	$\sum_{i=1}^{n} \left(\sum_{j=1}^{i} x_j\right)^2$	
Rastrigin	$\sum_{i=1}^{n} \left(x_i^2 - 10\cos(2\pi x_i) + 10\right)$	
Ackley	$-2e^{-0{,}2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}} - e^{\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)} + 20 + e$	
Griewank	$1 + \sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right)$	
Schaffer	$\left(x_1^2 + x_2^2\right)^{0{,}25} \left(\sin\left(50\left(x_1^2 + x_2^2\right)^{0{,}1}\right) + 1\right)$	

Tabelle 2.1.: Benchmarkfunktionen

Bei absolutem und relativem Rauschen ist die Varianz der Zufallsvariable konstant. Der Wert der Zufallsvariable wird bei absolutem Rauschen zum Funktionswert addiert und bei relativem als prozentuale Änderung interpretiert wird. Variables Rauschen entspricht absolutem Rauschen, wobei die Varianz der Zufallsvariable sich über den Faktorenraum hinweg ändert.

Teil I.

Optimierung

3. Response Surface Methode

3.1. Theorie der Response Surface Methode

3.1.1. Response Surface Methode – Was ist das?

Bartosz Fabianowski, Thomas Hutter

Die Response Surface Methode dient der Optimierung stochastischer Modelle. Der Begriff bezeichnet dabei jedoch nicht ein klar definiertes Verfahren, sondern steht für eine ganze Ansammlung unterschiedlicher mathematischer und stochastischer Techniken. Eine passendere Übersetzung für die englische Bezeichnung "Response Surface Methodology" ist daher "Response Surface Methodik". Die Ausgangslage ist in allen Fällen eine Zielfunktion, deren globales Extremum über einem Parameterraum gesucht wird. Die Werte der Zielfunktion werden, über den Parametern aufgetragen, als Fläche aufgefaßt (siehe Abb. 3.1). In Zusammenhang mit dieser Methodik werden die Parameter als Faktoren, die Fläche als Response Surface und die Zielfunktion als Response Surface Funktion bezeichnet.

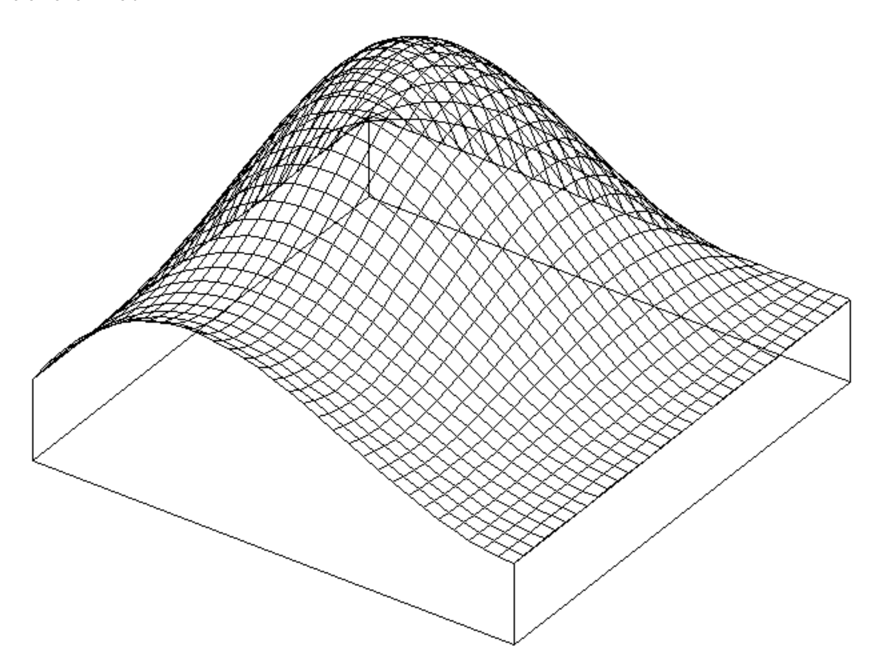


Abb. 3.1.: Response Surface

Diese Fläche wird nun von einem Startpunkt ausgehend iterativ in Richtung des vermuteten Optimums abgelaufen. Dazu sind in jeder Iteration die optimale Richtung und

Schrittweite zu bestimmen. Weil die Fläche im allgemeinen beliebig komplex sein kann und ihre Berechnung extrem aufwendig, wird eine lokale Approximation durch ein einfaches Polynom verwendet. Der Ablauf gliedert sich in vier Hauptschritte (siehe Abb. 3.2). Diese werden an dieser Stelle kurz vorgestellt und im Verlauf dieses Kapitels genauer erläutert.

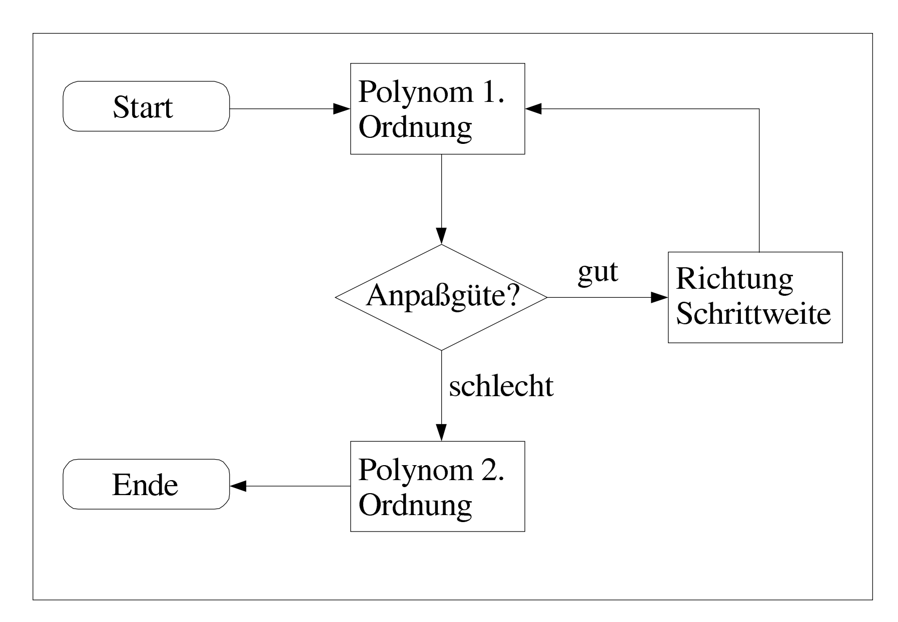


Abb. 3.2.: Ablauf Response Surface Methode

Zu Beginn des Verfahrens wird ein Startpunkt vorgegeben. Im ersten Schritt wird die Response Surface an der aktuell betrachteten Stelle lokal durch ein Polynom erster Ordnung approximiert. Im nächsten Schritt folgt ein Test der Anpaßgüte. Ist die Approximation ausreichend genau, dann werden in Schritt drei auf Basis des Polynoms eine Richtung sowie auf Basis der Response Surface Funktion eine Schrittweite bestimmt. Mit dem sich so ergebenden neuen Punkt wird das Verfahren iterativ wiederholt.

Ist die Approximation durch das Polynom erster Ordnung unzureichend, dann wird im Rahmen des vierten Schrittes die Response Surface durch ein Polynom zweiter Ordnung approximiert. Auf Grund des hohen damit verbundenen Aufwands wird diese Approximation im allgemeinen nur einmal durchgeführt und das Verfahren anschließend beendet.

Weil die Funktion jeweils nur in kleinen Bereichen lokal untersucht wird, führt die Response Surface Methode zum Auffinden lokaler Optima. Eine garantierte Bestimmung des globalen Optimums ist nicht möglich. Durch mehrfache Wiederholung mit unterschiedlichen Startpunkten werden dennoch oft gute Ergebnisse erzielt, so daß die Response Surface Methode vielfach eine Alternative zu anderen lokalen Optimierungsverfahren darstellt.

Bis heute wird dieser Ablauf zumeist manuell verfolgt. Dies liegt hauptsächlich daran, daß während seiner Durchführung einige nicht triviale Entscheidungen zu treffen sind, die sich nur schwer automatisieren lassen. So existieren für jeden Hauptschritt eine Fülle

von unterschiedlichen Verfahren, ohne daß quantitative Entscheidungskriterien zu ihrer Auswahl bekannt wären. Die Anwendung der Response Surface Methode stützt sich daher weitgehend auf menschliche Erfahrung und ein graduell aufgebautes Wissen über die zu optimierende Funktion.

Um den hohen damit verbundenen Zeitaufwand zu reduzieren, ist es heutzutage wünschenswert, die Anwendung der Methode zu automatisieren. Hierfür muß der Problembereich stark eingeschränkt werden, weil nur so die notwendigen Entscheidungen vorab getroffen und aus dem Verfahrensablauf eliminiert werden können. Dabei ist insbesondere zu berücksichtigen, daß ein Computer im Verlauf der Optimierung nicht die Response Surface Funktion immer besser verstehen und entsprechend reagieren kann.

Wir beschränken uns im Folgenden auf die Anwendung der Response Surface Methode im Bereich der stochastischen Simulation mit einer reellwertigen Response Surface Funktion. Auch möchten wir den Fall der linearen Optimierung ausschließen, weil hier der Simplexalgorithmus bereits global optimale Ergebnisse liefert.

3.1.2. Response Surface Methode für stochastische Simulationsmodelle

Bartosz Fabianowski, Thomas Hutter

Wir orientieren uns weitestgehend an dem Implementierungsvorschlag von Neddermeijer, Oortmarssen, Piersma und Dekker. Dieser Vorschlag aus dem Jahr 2000 beinhaltet eine detaillierte und begründete Auswahl von Verfahren für die einzelnen Schritte der Response Surface Methode. Abweichungen von diesem Implementierungsvorschlag sind im wesentlichen auf die Berücksichtigung diskreter Faktoren, die Festlegung konkreter maschinell auswertbarer Kriterien für alle zu treffenden Entscheidungen und die Optimierung des Ablaufs auf Basis von uns durchgeführter Leistungsstudien zurückzuführen.

3.1.2.1. Problemstellung

Die betrachtete Problemstellung sei folgende. Gegeben ist ein Simulationsmodell mit Faktorenraum $D \subseteq \mathbb{R}^k$ und stochastischer Zielfunktion $\mathrm{f} : D \to \mathbb{R}$. Gesucht ist ein Faktorenvektor $(\xi_1, \ldots, \xi_k) \in D$, für den der Erwartungswert $\mathrm{E}[\mathrm{f}]$ optimal wird. Unter optimal verstehen wir dabei o.B.d.A. minimal.

In jeder Iteration n wird die durch f festgelegte Response Surface in einem kleinen Bereich lokal betrachtet. Die Festlegung des Bereichs erfolgt dabei durch Angabe eines Mittelpunkts $(\xi_1^n, \ldots, \xi_k^n)$ und einer Ausdehnung $(c_1^n, \ldots, c_k^n)$. Der Bereich ist dann gegeben als:

$$[\xi_1^n - c_1^n, \xi_1^n + c_1^n] \times \ldots \times [\xi_k^n - c_k^n, \xi_k^n + c_k^n]$$

Die Approximation der Funktion durch ein Polynom findet jeweils nur innerhalb dieses Bereichs statt. Zur Vereinfachung der folgenden Betrachtung und um die numerischen Eigenschaften der Response Surface zu verbessern erfolgt eine Variablentransformation. Ein im lokalen Bereich liegender Faktorenvektor $(\xi_1, \ldots, \xi_k)$ wird durch einen Vektor $(x_1, \ldots, x_k)$ repräsentiert. Oft wird die Beziehung zwischen den beiden Vektoren so

gewählt, daß die x_i Mittelwert null und eine vorgegebene Varianz haben. Neddermeijer et al. schlagen folgende Verknüpfung vor:

$$x_i = \frac{\xi_i - \xi_i^n}{c_i^n} \Leftrightarrow \xi_i = c_i^n x_i + \xi_i^n$$

Es gilt $x_i \in [-1, 1]$ und $(x_1, \ldots, x_k)$ gibt die relative Position des Faktorenvektors $(\xi_1, \ldots, \xi_k)$ im aktuellen lokalen Bereich an. Die Response Surface Funktion wird nun ebenfalls in Abhängigkeit von $(x_1, \ldots, x_k)$ notiert:

$$\mathrm{f} : (x_1, \ldots, x_k) \to \mathbb{R}$$

3.1.2.2. Ablauffestlegung

Der Implementierungsvorschlag detailliert und erweitert das in Abbildung 3.2 dargestellte allgemeine Ablaufschema der Response Surface Methode. Das Verfahren gliedert sich in elf Schritte, die nachfolgend graphisch in den Gesamtzusammenhang eingeordnet und anschließend genau beschrieben werden.

3.1.2.2.A. Lokale Approximation der Response Surface Funktion durch ein Polynom 1. Ordnung

Die im allgemeinen beliebig komplexe Response Surface wird in diesem ersten Schritt durch eine einfache Hyperebene approximiert. Eine solche Ebene kann vollständig beschrieben werden durch ein Polynom 1. Ordnung. In Abhängigkeit vom ursprünglichen Faktorenvektor $(\xi_1, \ldots, \xi_k)$ hat dieses die Form:

$$y = \alpha_0 + \sum_{i=1}^{k} \alpha_i \xi_i$$

Ausgedrückt in Abhängigkeit vom transformierten Vektor $(x_1, \ldots, x_k)$ ergibt sich entsprechend:

$$y = \beta_0 + \sum_{i=1}^{k} \beta_i x_i$$

Die Regressionskoeffizienten β_i sind so zu wählen, daß die Hyperebene die Response Surface im lokalen Bereich möglichst gut approximiert. Weil die Gleichung der Fläche nicht in geschlossener Form vorliegt, kann sie nur an einer endlichen Zahl Stützstellen ausgewertet werden. Zusätzlich ist eine Auswertung gleichbedeutend mit der Durchführung eines Simulationslaufs, womit sie sehr kostspielig und aufwendig ist. Ziel dieses Schrittes ist es daher, mit möglichst wenigen Auswertungen eine möglichst gute Approximation für die Regressionskoeffizienten zu berechnen. Die Auswahl der zu verwendenden Stützstellen wird mit dem englischen Begriff Design of Experiment bezeichnet. Neddermeijer et al. schlagen vor, ein Resolution-III Two-Level Fractional Factorial Design zu

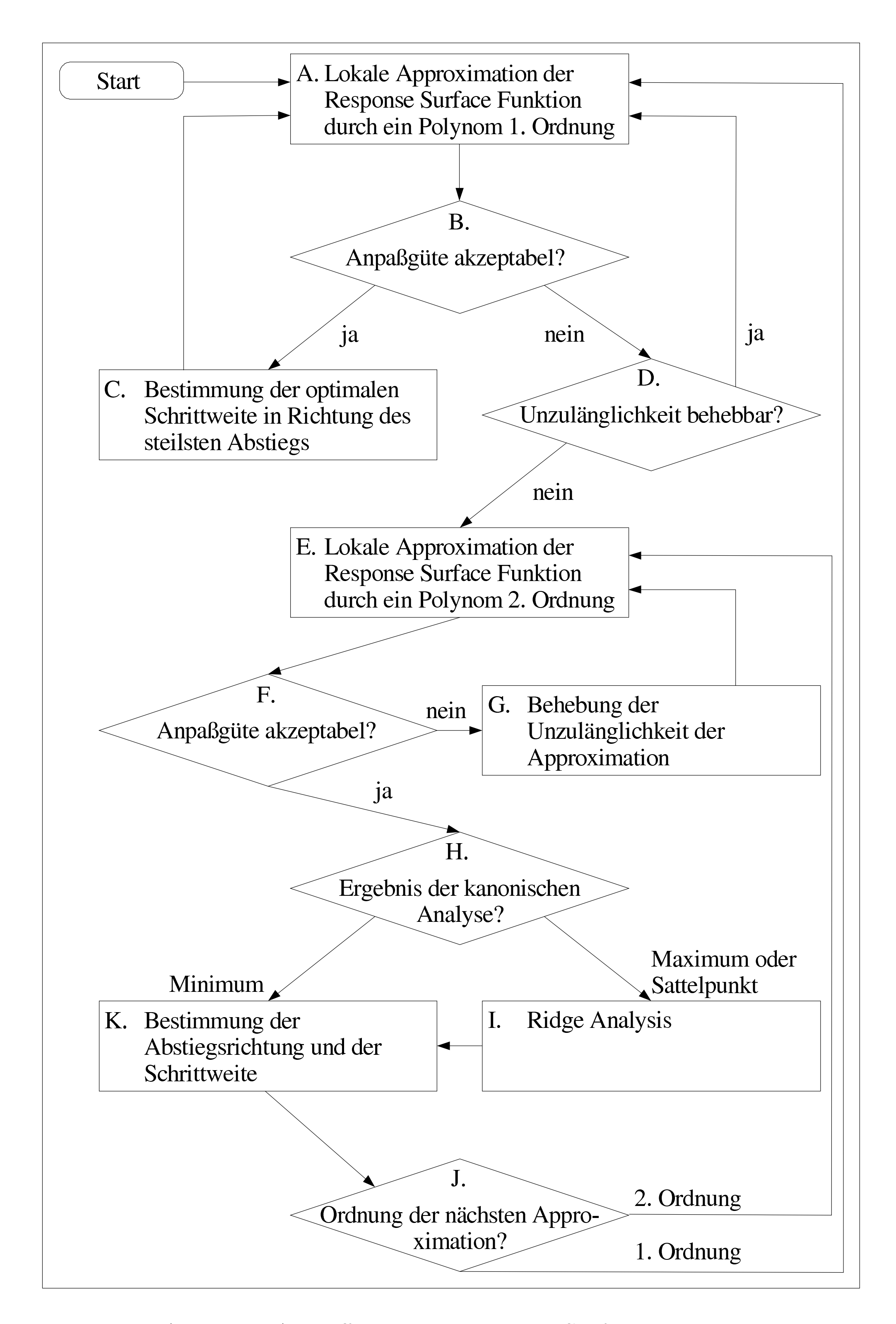


Abb. 3.3.: Ablauffestlegung Response Surface Methode

verwenden. Für weitere Informationen zur Bedeutung und zum Aufbau eines solchen Designs verweisen wir an dieser Stelle auf das Kapitel 3.1.3.

Durch Wahl eines den Kriterien entsprechenden Designs werden die Stützstellen festgelegt. Für diskrete Faktoren runden wir die Stützstellen anschließend auf ganzzahlige Faktorenwerte. Obwohl die theoretische Korrektheit dieses Vorgehens nicht erwiesen ist, hat es in unseren Leistungsstudien zu unserer Zufriedenheit funktioniert.

Zur Ermittlung der zugehörigen Werte der Response Surface Funktion wird für jede Stützstelle ein Simulationslauf durchgeführt. Das Ergebnis dieser Auswertung ist für jede $i$te Stützstelle mit einem Faktorenvektor $(x_{i1}, \ldots, x_{ik})$ ein Funktionswert y_i.

Die letzte verbleibende Aufgabe in diesem Schritt der Response Surface Methode ist die lokale Approximation der Response Surface durch eine Hyperebene auf Basis der berechneten Funktionswerte. Wie zuvor gezeigt wird die Hyperebene durch ein Polynom 1. Ordnung beschrieben:

$$y = \beta_0 + \sum_{i=1}^{k} \beta_i x_i$$

Die Abweichung der Response Surface von der Hyperebene an der $i$ten Stützstelle wird durch einen Fehlerterm ε_i ausgedrückt:

$$\varepsilon_i = y_i - y \;\Leftrightarrow\; \varepsilon_i = y_i - \beta_0 - \sum_{j=1}^{k} \beta_j x_{ij}$$

Es wird angenommen, daß die Fehlerterme alle unabhängig und identisch verteilt mit Erwartungswert null und Varianz σ^2 sind.

Die gängige Methode zur Anpassung der Hyperebene an die Response Surface wird als Least Squares bezeichnet. Dabei wird versucht, die Summe der quadrierten Fehlerterme ε_i an den Stützstellen zu minimieren:

$$L = \sum_{i=1}^{n} \varepsilon_i^2 = \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{k} \beta_j x_{ij} \right)^2$$

Die notwendige Bedingung für das Vorliegen eines Extremums ist $\operatorname{grad} L = 0$. Gesucht ist also eine Belegung $(b_0, \ldots, b_k)$ für die Regressionskoeffizienten $(\beta_0, \ldots, \beta_k)$, so daß gilt:

$$\begin{aligned} \left.\frac{\partial L}{\partial \beta_0}\right|_{b_0,\ldots,b_k} &= -2 \sum_{i=1}^{n} \left(y_i - b_0 - \sum_{j=1}^{k} b_j x_{ij} \right) = 0 \\ \wedge\; \forall 1 \le j \le k: \; \left.\frac{\partial L}{\partial \beta_j}\right|_{b_0,\ldots,b_k} &= -2 \sum_{i=1}^{n} \left(y_i - b_0 - \sum_{j=1}^{k} b_j x_{ij} \right) x_{ij} = 0 \end{aligned}$$

Dieses Gleichungssystem kann in der vorliegenden Form gelöst werden, allerdings ergibt sich eine Fülle von komplizierten Termen, die nur schwer zu handhaben sind. Einfacher ist die systematische Betrachtung des Gleichungssystems für alle Stützstellen in Matrixform:

$$\vec{Y} = X\vec{\beta} + \vec{\varepsilon}$$

Dabei sind die Vektoren und die Matrix gegeben als:

$$\vec{Y} = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}, \quad X = \begin{pmatrix} 1 & x_{11} & \cdots & x_{1k} \\ \vdots & \vdots & & \vdots \\ 1 & x_{n1} & \cdots & x_{nk} \end{pmatrix}, \quad \vec{\beta} = \begin{pmatrix} \beta_0 \\ \vdots \\ \beta_k \end{pmatrix}, \quad \vec{\varepsilon} = \begin{pmatrix} \varepsilon_1 \\ \vdots \\ \varepsilon_n \end{pmatrix}$$

Für die zu minimierende Summe der quadrierten Fehlerterme ergibt sich:

$$L = \sum_{i=1}^{n} \varepsilon_i^2 = \vec{\varepsilon}^T \cdot \vec{\varepsilon} = \left(\vec{Y} - X\vec{\beta}\right)^T \left(\vec{Y} - X\vec{\beta}\right)$$

Die notwendige Bedingung für das Vorliegen eines Extremums, $\operatorname{grad} L = 0$, wird damit zu:

$$\left.\frac{\partial L}{\partial \vec{\beta}}\right|_{\vec{b}} = -2X^T\vec{Y} + 2X^TX\vec{b} = 0$$

Die gesuchte Belegung $\vec{b}$ kann durch einfaches Umformen der Gleichung ermittelt werden:

$$\vec{b} = \left(X^TX\right)^{-1} X^T\vec{Y}$$

Damit ist die Aufgabe von Schritt A abgeschlossen. Die Approximationsebene ist gegeben durch:

$$\hat{y} = b_0 + \sum_{j=1}^{k} b_j x_{ij}$$

3.1.2.2.B. Anpaßgüte akzeptabel?

Es stellt sich die Frage, ob das im vorhergehenden Abschnitt berechnete Polynom die Response Surface Funktion in angemessener Weise approximiert. Die Beantwortung dieser Frage ist wichtig, weil im nächsten Schritt die Richtung des steilsten Abstiegs auf Basis des Polynoms bestimmt werden soll. Wird die Funktion durch das Polynom nur unzureichend angenähert, dann kann die völlig falsche Richtung eingeschlagen werden, womit das Verfahren sich nicht auf das vermutete Optimum zu, sondern von ihm hinweg bewegt. Zugleich ist die Beantwortung der Frage auch äußerst schwierig, weil über die Response Surface und ihre definierende Funktion nur sehr wenig bekannt ist. Eine Fläche mit starker Krümmung etwa kann auch lokal durch eine Hyperebene nur sehr schlecht

approximiert werden. Weil das Krümmungsverhalten der Response Surface jedoch im allgemeinen unbekannt ist, muß ein universelles Kriterium gefunden werden, das zudem automatisch ausgewertet werden kann.

Notwendig ist ein Testverfahren, bei dem eine Teststatistik aufgestellt und mit den kritischen Werten einer zugehörigen Verteilung verglichen werden kann. Das gängige Verfahren basiert auf der Analyse der Varianz der Fehlerterme ε_i, um die Abweichung der Hyperebene von der Response Surface Funktion abzuschätzen. Es wird mit ANOVA, kurz für Analysis of Variance, bezeichnet. Die Berechnung der Teststatistik und ihr Vergleich mit der F-Verteilung sind ohne manuelle Eingriffe durchführbar, womit sich das Verfahren für die angestrebte Automatisierung sehr gut eignet. Um zwischen zufälligen Abweichungen und der tatsächlichen Abweichung der Hyperebene von der Response Surface unterscheiden zu können, benötigt der ANOVA Test mehrere Funktionswerte pro Stützstelle. Es sind also weitere Simulationsläufe durchzuführen. Dabei müssen nicht für alle im vorhergehenden Schritt verwendeten Stützstellen zusätzliche Funktionswerte bestimmt werden. Es genügt, nur einige wenige Stellen zu betrachten. Die Differenz zwischen jedem per Simulation ermittelten Funktionswert y_i und dem zugehörigen Wert des Approximationspolynoms $\hat{y}_i$ wird als Residuum e_i bezeichnet:

$$e_i = y_i - \hat{y}_i$$

Die Summe der quadrierten Residuen wird SS_E genannt. Mit $p = k + 1$ der Anzahl der geschätzten Regressionskoeffizienten hat sie $n - p$ Freiheitsgrade und berechnet sich zu:

$$SS_E = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

Die Beobachtungen werden nun nach den zugehörigen Stützstellen gruppiert. Wenn bei insgesamt m verwendeten Stützstellen für die $i$te Stelle n_i Simulationsläufe durchgeführt wurden, dann werden ihre Ergebnisse als y_{ij} mit $1 \leq j \leq n_i$ bezeichnet. Damit kann SS_E geschrieben werden als:

$$SS_E = \sum_{i=1}^{m} \sum_{j=1}^{n_i} (y_{ij} - \hat{y}_i)^2$$

Für jede $i$te Stützstelle errechnet der Durchschnitt der Beobachtungen $\overline{y}_i$ sich zu:

$$\overline{y}_i = \frac{\sum_{j=1}^{n_i} y_{ij}}{n_i}$$

Ein Residuum $e_{ij} = y_{ij} - \hat{y}_i$ kann damit zerlegt werden in:

$$y_{ij} - \hat{y}_i = (y_{ij} - \overline{y}_i) + (\overline{y}_i - \hat{y}_i)$$

Wird diese Zerlegung auf alle Summenterme von SS_E angewandt, ergibt sich:

$$\sum_{i=1}^{m}\sum_{j=1}^{n_i}(y_{ij}-\hat{y}_i)^2=\sum_{i=1}^{m}\sum_{j=1}^{n_i}(y_{ij}-\overline{y}_i)^2+\sum_{i=1}^{m}n_i\,(\overline{y}_i-\hat{y}_i)^2$$

Die verbleibende Doppelsumme wird unter der Bezeichnung SS_{PE} geführt. Ihr Wert gibt die Summe der quadrierten "echten" Fehler an, also der Abweichungen der Beobachtungen an den einzelnen Stützstellen von den zugehörigen Mittelwerten durch stochastische Schwankungen. SS_{PE} hat $n-m$ Freiheitsgrade, weil bei insgesamt n Funktionsauswertungen m Mittelwerte berechnet werden. Der mittlere quadrierte echte Fehler MS_{PE} ist somit:

$$MS_{PE}=\frac{SS_{PE}}{n-m}$$

Die Einfachsumme, bezeichnet mit SS_{LOF}, ist die Summe der quadrierten Abweichungen des Approximationspolynoms von den mittleren Beobachtungen an den einzelnen Stützstellen. Die Anzahl der Freiheitsgrade beträgt $m-p$, weil aus den an m Stützstellen ermittelten Werten p Parameter geschätzt werden. Die mittlere quadrierte Abweichung des Approximationspolynoms MS_{LOF} ist gegeben als:

$$MS_{LOF}=\frac{SS_{LOF}}{m-p}$$

Die ANOVA Teststatistik F_0 ist nun das Verhältnis von MS_{LOF} zu MS_{PE} :

$$F_0=\frac{MS_{LOF}}{MS_{PE}}$$

Ein großer Wert von F_0 bedeutet, daß die Abweichungen durch die Approximation im Verhältnis zu den stochastischen Schwankungen der Response Surface Funktion nicht vernachlässigbar sind. Das gefundene Polynom 1. Ordnung stellt also eine schlechte Approximation dar. Im Gegensatz dazu bedeutet ein kleiner Wert, daß die Abweichungen durch die Approximation gegenüber den stochastischen Schwankungen gering sind. In diesem Falle ist die Approximation gut. Der konkrete kritische Wert für F_0 zum gewünschten Signifikanzniveau kann aus der $F_{m-n,n-p}$-Verteilung, die als Quotient zweier χ^2-Verteilungen definiert ist, berechnet werden.

Die zur Bestimmung der Teststatistik benötigten Werte werden üblicherweise in einer Tabelle zusammengefaßt. Diese kann auch zur Durchführung weitergehender Tests verwendet werden, wobei sie dann um zusätzliche Zeilen erweitert wird. Wir geben an dieser Stelle nur die für den oben angeführten Test benötigten Zeilen an (Tabelle 3.1).

Wird im Ergebnis des Tests die Approximation durch das Polynom 1. Ordnung akzeptiert, dann folgt gemäß dem Ablaufschema Schritt C, in dem auf Basis des Polynoms eine Abstiegsrichtung berechnet wird. Sollte das Polynom dagegen die Response Surface Funktion nicht adäquat approximieren, dann wird mit Schritt D fortgefahren, in dem versucht wird, die Bestimmung eines Polynoms von akzeptabler Qualität zu ermöglichen.

Quelle der Varianz	Summe der Quadrate	Freiheitsgrade	Mittleres Quadrat	F_0
Residuum	SS_E	$n-p$	MS_E	–
Approximation	SS_{LOF}	$m-p$	MS_{LOF}	$\frac{MS_{LOF}}{MS_{PE}}$
Echter Fehler	SS_{PE}	$n-m$	MS_{PE}	–

Tabelle 3.1.: ANOVA Tabelle

3.1.2.2.C. Bestimmung der optimalen Schrittweite in Richtung des steilsten Abstiegs

Dieser Schritt gliedert sich in zwei Teile – die Bestimmung einer Abstiegsrichtung sowie die anschließende Berechnung der Schrittweite. ir betrachten zunächst die erste Aufgabenstellung, die Berechnung der Richtung des steilsten Abstiegs an Hand eines Polynoms 1. Ordnung. Mit $(b_0, \ldots, b_k)$ der Belegung der Regressionskoeffizienten des Polynoms ist diese Richtung gegeben als:

$$(-b_1, \ldots, -b_k)$$

Die Koeffizienten geben jeweils die Steigung des Polynoms gegenüber den einzelnen Faktoren an. In entgegengesetzter Richtung zu dieser Steigung findet der steilste Abstieg statt. Die Bestimmung der Abstiegsrichtung ist damit trivial und auf einfache Weise nachvollziehbar. Dennoch ist bei Bedarf auch eine mathematische Herleitung der Abstiegsrichtung möglich, die zum obigen Ergebnis führt.

Der erste Teil von Schritt C ist damit abgeschlossen. Im zweiten Teil besteht die Aufgabe darin, in der gewählten Abstiegsrichtung eine Schrittweite zu bestimmen. Dabei sollen so lange Schritte in Richtung des steilsten Abstiegs durchgeführt werden, wie der Funktionswert sich dadurch verbessert. Sobald der Funktionswert beginnt, sich wieder zu verschlechtern, wird der Abstieg beendet. Dies ist eine lineare Suche nach dem nächstgelegenen lokalen Minimum.

Um Schritte durchführen zu können, muß neben ihrer Richtung, die durch den Vektor $(-b_1, \ldots, -b_k)$ vorgegeben ist, auch ihre Größe festgelegt werden. Dazu ist für jeden Faktor zu bestimmen, um wie viel er sich pro Schritt ändert. Die Schrittgröße kann folglich ebenfalls als Vektor geschrieben werden:

$$(\Delta_1, \ldots, \Delta_k)$$

Das Verhältnis der Komponenten dieses Vektors zueinander ist durch die Abstiegsrichtung festgelegt:

$$\Delta_1 \div \ldots \div \Delta_k = b_1 \div \ldots \div b_k$$

Damit wird die Schrittgröße durch Skalierung des Vektors $(-b_1, \ldots, -b_k)$ gebildet. Die einzige frei wählbare Größe ist der Skalierungsfaktor. Die übliche Methode zu seiner

Festlegung sieht vor, daß zunächst ein Faktor ξ_j subjektiv als wichtigster hervorgehoben wird. Bei Implementierung der Response Surface Methode ist eine solche subjektive Entscheidungsfindung während der Optimierung nicht wünschenswert. Die erste Alternative besteht darin, die Entscheidung vor Beginn der Optimierung einmalig subjektiv zu treffen. Um auch diesen manuellen Eingriff zu eliminieren, ist eine vollständig automatische Bestimmung des wichtigsten Faktors ebenfalls möglich. Als wichtigster Faktor wird in diesem Fall derjenige gewählt, dessen zugehöriger Regressionskoeffizient den größten Absolutwert hat:

$$j = \arg\max_{i=1,\ldots,k} |b_i|$$

Als Skalierungsfaktor zur Bestimmung der Schrittgröße wird anschließend $|b_j|$ verwendet, so daß sich für die einzelnen Komponenten der Größe ergibt:

$$\forall 1 \leq i \leq k: \;\; \Delta_i = \frac{-b_i}{|b_j|}$$

Für den wichtigsten Faktor ξ_j errechnet sich damit $\Delta_j = -1$. Mit dieser Schrittgröße findet nun eine lineare Suche statt. Ausgehend vom aktuellen Faktorenvektor ist der nach dem $m$ten Schritt betrachtete Punkt:

$$\{\xi_1^n + m\Delta_1 c_1^n, \ldots, \xi_k^n + m\Delta_k c_k^n\}$$

c_i^n gibt jeweils die Entfernung vom Mittelpunkt zum Rand des aktuellen lokalen Bereichs bei Veränderung des Faktors ξ_i an. Die festgelegte Schrittgröße stellt daher wegen $\Delta_j = -1$ sicher, daß für den wichtigsten Faktor ξ_j der Rand des lokalen Bereichs nach genau einem Schritt erreicht wird. Mit jedem weiteren Schritt wird nochmal dieselbe Entfernung zurückgelegt. Die Veränderungen der anderen Faktoren ergeben sich proportional dazu. Sollte bei einem Schritt der zulässige Suchbereich verlassen werden, dann wird die Suche entlang der Projektion der Abstiegsrichtung auf den Rand des Suchbereichs fortgesetzt.

Es muß nun noch festgestellt werden, wann der Funktionswert der Response Surface Funktion beginnt, sich wieder zu verschlechtern und die Suche beendet werden soll. Dies geschieht durch die Festlegung eines Stopkriteriums. An dieser Stelle sollte hervorgehoben werden, daß ein Stopkriterium nur an Hand der Response Surface Funktion selber und nicht auf Basis des Approximationspolynoms angegeben werden kann. Denn ein Polynom 1. Ordnung etwa fällt in Richtung des steilsten Abstiegs kontinuierlich ab und könnte demnach nie zu einer Terminierung der Suche führen.

Das einfachste Stopkriterium besteht darin, die Suche abzubrechen, sobald ein errechneter Funktionswert höher ist, als der unmittelbar vorhergehende. Dieses Kriterium hat jedoch den gravierenden Nachteil, daß es von den stochastischen Schwankungen der Response Surface Funktion stark beeinflußt wird. Denn jede Auswertung der Funktion geschieht durch Simulation und ist damit ein Zufallsexperiment, das zu einem der vielen möglichen Ergebnisse führt. Ein Faktorenvektor, der im Mittel zu einem geringen Funktionswert führt, kann zufällig in einigen Experimenten auch zu einem hohen Wert führen, womit die Suche zu früh abgebrochen wird. Ebenso ist umgekehrt ein versehentliches

Überschreiten des gesuchten Punktes möglich. Wir brechen daher die Suche erst dann ab, wenn eine zuvor festgelegte Anzahl aufeinander folgender Punkte einen schlechteren Funktionswert aufweisen als ihr jeweiliger Vorgänger.

Sobald eine Schrittweite ermittelt worden ist, wird der Schritt vollzogen. Gemäß dem Schema wird das Verfahren mit dem neuen Faktorenvektor als Mittelpunkt des lokalen Bereichs iterativ wiederholt. Der in Abb. 3.2 dargestellte Kreislauf im Grobschema der Response Surface Methode ist damit vollständig erläutert.

3.1.2.2.D. Unzulänglichkeit behebbar?

Wie in Punkt C dargestellt, kann die Approximation für unzureichend befunden werden. Je nach Umfang der Unzulänglichkeit existieren verschiedene Methoden für ihre Behebung. In den meisten Fällen wird, wie im Grobschema (Abb. 3.2) angedeutet, auf eine Approximation durch ein Polynom 2. Ordnung ausgewichen. Während ein solches Polynom eine höhere Anpaßgüte verspricht, ist seine Berechnung jedoch weitaus aufwendiger und kostspieliger. Daher ist es wünschenswert, die Approximation durch ein Polynom 1. Ordnung zu verbessern.

Wurde das aktuelle Approximationspolynom nur knapp verworfen, dann versuchen wir deswegen, ein Polynom 1. Ordnung mit besserer Anpaßgüte zu bestimmen. Dazu erhöhen wir die Anzahl der Auswertungen je Stützstelle und kehren zur Bestimmung eines Approximationspolynoms in Schritt A zurück. Die Mittelung mehrerer Funktionswerte pro Stützstelle kann ihre stochastischen Schwankungen verringern und so eine gute Approximation erleichtern.

Hat sich das Polynom 1. Ordnung dagegen als deutlich unzureichend erwiesen, dann Fahren wir mit der Approximation durch ein Polynom 2. Ordnung in Schritt E fort.

3.1.2.2.E. Lokale Approximation der Response Surface Funktion durch ein Polynom 2. Ordnung

Kann die Qualität der Approximation durch ein Polynom 1. Ordnung auch durch Korrekturmaßnahmen nicht auf ein akzeptables Niveau erhöht werden, dann erfolgt in diesem Schritt eine Approximation unter Verwendung eines Polynoms 2. Ordnung. Der Ablauf des Schrittes gliedert sich analog zu Schritt A in drei Teile. Zunächst wird ein Design of Experiment bestimmt, es werden also die Stützstellen festgelegt, an denen die Response Surface Funktion auszuwerten ist. Anschließend wird die Auswertung durch Simulation durchgeführt. Im dritten Teil erfolgt dann die Bestimmung der Regressionskoeffizienten eines Polynoms 2. Ordnung.

Für das Design of Experiment verwenden wir dem Vorschlang von Neddermeijer et al. folgend ein Central Composite Design. Zum Aufbau dieses Designs verweisen wir wieder auf das Kapitel 3.1.3 zum Thema Experimental Design.

Nach Wahl eines Designs ist im zweiten Teil von Schritt E die Response Surface Funktion an den Stützstellen auszuwerten. Die Anzahl der Funktionsauswertungen kann bei Wahl eines entsprechenden Designs durch Wiederverwendung von in Schritt A ermit-

telten Werten verringert werden, bleibt jedoch in jedem Fall nicht unerheblich. Jede Auswertung entspricht der Durchführung eines vollständigen Simulationslaufs, womit dies die zeitaufwendigste Aufgabe ist.

Im dritten und letzten Teil schließlich wird ein Polynom 2. Ordnung an die ermittelten Funktionswerte angepaßt. Ein solches Polynom hat in Abhängigkeit vom transformierten Faktorenvektor $(x_1, \dots, x_k)$ die Form:

$$y = \beta_0 + \sum_{i=1}^{k} \beta_i x_i + \sum_{i=1}^{k} \sum_{j=1}^{k} \beta_{i,j} x_i x_j$$

Zur Ermittlung einer Belegung für die Regressionskoeffizienten wird, wie in Schritt A, das Least Squares Verfahren verwendet. Analog zum Vorgehen in jenem Schritt wird dabei die Summe der quadrierten Abweichungen zwischen Approximationspolynom und Funktionswert an den Stützstellen minimiert. Wir verzichten an dieser Stelle darauf, das Verfahren ein weiteres Mal darzustellen und verweisen diesbezüglich auf unsere Ausführungen zu Schritt A. Das Ergebnis ist eine Belegung b_i für jedes β_i und $b_{i,j}$ für jedes $\beta_{i,j}$, so daß das Approximationspolynom die Form hat:

$$\hat{y} = b_0 + \sum_{i=1}^{k} b_i x_i + \sum_{i=1}^{k} \sum_{j=1}^{k} b_{i,j} x_i x_j$$

3.1.2.2.F. Anpaßgüte akzeptabel?

Analog zu Schritt B ist in diesem Schritt zu überprüfen, ob das soeben berechnete Approximationspolynom die Response Surface Funktion im aktuellen lokalen Bereich ausreichend genau approximiert. Dazu können dieselben Testmethoden herangezogen werden. Es kommt also wieder die ANOVA Tabelle zum Einsatz.

Wie in Schritt B wird je nachdem, ob die Anpaßgüte des Polynoms für ausreichend befunden wird oder nicht, zu unterschiedlichen Schritten verzweigt. Ist die Qualität der Approximation akzeptabel, dann wird das Polynom in Schritt H untersucht, um die Richtung des steilsten Abstiegs zu bestimmen. Ist sie nicht gut genug, dann wird in Schritt G versucht, die Voraussetzungen für eine bessere Approximation zu schaffen.

3.1.2.2.G. Behebung der Unzulänglichkeit der Approximation

Wird das gefundene Approximationspolynom 2. Ordnung in Schritt F als unzureichend verworfen, dann ist es Ziel dieses Schrittes, die Güte der Approximation zu erhöhen. Anders als in Schritt D besteht nicht die Möglichkeit, auf die Approximation mittels eines Polynoms höherer Ordnung auszuweichen. Denn zur Bestimmung eines Approximationspolynoms 3. Ordnung wären so unverhältnismäßig viele Auswertungen der Response Surface Funktion notwendig, daß sie in RSM üblicherweise und auch in unserer Implementierung nicht eingesetzt wird.

Wir versuchen daher immer, durch Erhöhung der Anzahl der Auswertungen pro Stützstelle günstigere Bedingungen für eine Approximation mit einem Polynom 2. Ordnung zu schaffen. Dazu kehren wir im Ablauf wieder zu Schritt E zurück. Ein solches Verhalten könnte leicht zu einer endlosen Wiederholung von Approximation und Korrektur führen. Sobald die Anzahl der Auswertungen je Stützstelle einen vorgegebenen Grenzwert erreicht hat, brechen wir daher stattdessen die Optimierung ab.

3.1.2.2.H. Ergebnis der kanonischen Analyse?

Die Bestimmung der besten Abstiegsrichtung an Hand eines Polynoms 2. Ordnung ist nicht trivial. Sie ist in der Ablauffestlegung der Response Surface Methode aus Abb. 3.3 daher in mehrere Schritte gegliedert, die jeweils für unterschiedliche Fälle zum Einsatz kommen. Die kanonische Analyse stellt den ersten Teil der Auswertung dar, in der über das weitere Vorgehen entschieden wird. Sie beginnt mit der Ermittlung eines stationären Punkts des Polynoms. Die Entscheidung über den weiteren Ablauf der Analyse wird dann auf Basis der Lage und Art des Punktes getroffen. Ein stationärer Punkt liegt dort vor, wo die erste Ableitung des Polynoms nach dem transformierten Faktorenvektor $(x_1, \ldots, x_k)$ null wird. Diese Ableitung läßt sich am einfachsten bestimmen, wenn das Approximationspolynom in Matrixschreibweise notiert wird:

$$\hat{y} = b_0 + \vec{X}^T \vec{b} + \vec{X}^T B \vec{X}$$

Dabei sind die Vektoren und die symmetrische Matrix B gegeben als:

$$\vec{X} = \begin{pmatrix} x_1 \\ \vdots \\ x_k \end{pmatrix}, \quad B = \begin{pmatrix} b_{1,1} & b_{1,2}/2 & \cdots & b_{1,k}/2 \\ b_{1,2}/2 & b_{2,2} & \cdots & b_{2,k}/2 \\ \vdots & \vdots & \ddots & \vdots \\ b_{1,k}/2 & b_{2,k}/2 & \cdots & b_{k,k} \end{pmatrix}, \quad \vec{b} = \begin{pmatrix} b_1 \\ \vdots \\ b_k \end{pmatrix}$$

Die erste Ableitung des Polynoms nach $\vec{X}$ ist nun:

$$\frac{\partial \hat{y}}{\partial \vec{X}} = \vec{b} + 2B\vec{X}$$

Wird diese Ableitung zu null gesetzt und nach $\vec{X}$ aufgelöst, so ergibt sich $\vec{X}_s$, der stationäre Punkt:

$$\vec{X}_s = -\frac{1}{2} B^{-1} \vec{b}$$

Mit diesem Punkt beginnt die eigentliche kanonische Analyse. Zuerst wird die Translation $\vec{Z} = \vec{X} - \vec{X}_s$ durchgeführt, so daß der stationäre Punkt auf den Ursprung abgebildet wird. Für das Approximationspolynom folgt durch Einsetzen und Umformen:

$$\hat{y} = \hat{y}_s + \vec{Z}^T B \vec{Z}$$

Als nächstes erfolgt eine Rotation. Dazu wird aus den Eigenwerten λ_i der Matrix B eine Diagonalmatrix Λ gebildet. Die zugehörigen normalisierten Eigenvektoren formen die Matrix P. Für die Matrizen gilt die Beziehung:

$$P^T B P = \Lambda$$

Die Rotation wird nun mit $\vec{W} = P^T \vec{Z}$ durchgeführt. Dadurch werden die Hauptachsen des Approximationspolynoms auf die Achsen des Koordinatensystems abgebildet. Nach Einsetzen und Umformen ergibt sich:

$$\hat{y} = \hat{y}_s + \vec{W}^T \Lambda \vec{W}$$

Mit w_i den Elementen des Vektors $\vec{W}$ kann die Gleichung auch geschrieben werden als:

$$\hat{y} = \hat{y}_s + \sum_{i=1}^{k} \lambda_i w_i^2$$

Diese Darstellung des Polynoms wird als kanonische Form bezeichnet, die w_i als kanonische Variablen. Weil durch die Transformationen der eigentliche Funktionswert $\hat{y}$ nicht verändert wurde, kann die kanonische Form dazu genutzt werden, Aussagen über das ursprüngliche Polynom zu treffen.

Sind alle λ_i positiv, dann hat das Polynom im stationären Punkt $\vec{X}_s$ ein Minimum mit dem zugehörigen Funktionswert $\hat{y}_s$. Sind sie im Gegensatz dazu alle negativ, dann liegt ein Maximum vor. Bei gemischten Vorzeichen der λ_i handelt es sich um einen Sattelpunkt.

Handelt es sich bei dem stationären Punkt $\vec{X}_s$ um ein Minimum, dann ist es im Rahmen der Response Surface Methode interessant, die Response Surface Funktion in Richtung dieses Minimums zu verfolgen. In diesem Fall wird der Ablauf mit Schritt K fortgesetzt, in dem die genaue Abstiegsrichtung berechnet wird. Neddermeijer et al. schlagen an dieser Stelle eine Sonderbehandlung für Minima außerhalb des aktuellen lokalen Bereichs vor. Es könnte interessant sein, zu solchen Punkten zu direkt hinspringen, um so die rein lokale Optimierung auf weit entfernte vielversprechende Regionen auszuweiten. Im Interesse einer stabilen Optimierung ohne Sprünge zu völlig neuen Faktorbelegungen haben wir auf diese Möglichkeit verzichtet.

Wurde ein Maximum oder ein Sattelpunkt ermittelt, dann wird mit der Ridge Analysis in Schritt I fortgefahren, die durch Einführen einer Nebenbedingung die Bestimmung eines Minimums erzwingen kann.

3.1.2.2.I. Ridge Analysis

Die Ridge Analysis kann das Auffinden eines Minimums des Approximationspolynoms erzwingen, indem sie eine Nebenbedingung einführt. Diese hat die Form $\vec{X}_{S,R}^T \vec{X}_{S,R} = R^2$

mit $\vec{X}_{S,R}$ dem zu bestimmenden Minimum und R einer frei wählbaren Konstante. Gesucht wird nun also das Minimum des Polynoms auf einer Hyperkugel um den Mittelpunkt des aktuellen lokalen Bereichs mit dem Radius R. Der Radius sollte so gewählt werden, daß die gesamte Kugel innerhalb des lokalen Bereichs liegt. Falls also etwa für den Bereich $[-1,1]^k$ gilt, dann darf R höchstens den Wert $\sqrt{2}$ annehmen. Zu lösen ist nun also das folgende restringierte Optimierungsproblem:

$$\begin{array}{ll} \min & b_0 + \vec{X}_R^T \vec{b} + \vec{X}_R^T B \vec{X}_R \\ \text{udN} & \vec{X}_R^T \vec{X}_R = R^2 \end{array}$$

Zur Lösung dieses quadratischen Optimierungsproblems mit quadratischen Nebenbedingungen haben wir die im Rahmen der Ridge Analysis ursprünglich vorgesehene Methode verwendet. Diese führt zunächst Lagrange'sche Multiplikatoren μ_i ein und transformiert das Problem anschließend unter Verwendung der Eigenwerte der Matrix B in ein numerisches Problem.

Während unserer Leistungsstudien sind vereinzelte Fälle aufgetreten, in denen dieses Verfahren sich aufhing und nicht terminierte. Auch zeigte sich, daß es bei diskreten Faktoren grundsätzlich versagt. Weil wir die gesamte Herleitung des numerischen Lösungsverfahrens nicht kennen, konnten wir die Quelle des Problems nicht genau ermitteln. Wir haben die Ridge Analysis daher aus dem Ablauf der Optimierung entfernt und gehen stattdessen direkt zu Schritt K über, um eine lineare Suche ohne die Berücksichtigung eines Ridge Minimums durchzuführen.

3.1.2.2.J. Ordnung der nächsten Approximation?

In diesem Schritt entscheidet sich, ob für die nächste Iteration ein Polynom 1. oder 2. Ordnung zur Approximation der Response Surface Funktion verwendet wird. Beim ursprünglichen Vorschlag von Neddermeijer et al. sind Sprünge zu weit entfernten Minima des Approximationspolynoms möglich. Es ist daher stets eine Abwägung durchzuführen zwischen der effizienten Approximation mittels eines Polynoms 1. Ordnung und der weitaus kostspieligeren Verwendung eines Polynoms 2. Ordnung, die dafür solche Sprünge ermöglicht.

Weil wir darauf verzichten, Sprünge im Faktorenraum durchzuführen, schreitet unsere Optimierung nur durch lineare Suchen fort. Bei Erreichen einer neuen besten Faktorenbelegung müssen wir die umliegende Response Surface daher nur soweit kennenlernen, wie es für die Durchführung der nächsten linearen Suche notwendig ist. Wir beginnen deswegen stets mit einer Approximation durch ein Polynom 1. Ordnung. Sollte diese sich als nicht ausreichend erweisen, kann im Nachhinein die Bestimmung eines Approximationspolynoms 2. Ordnung durchgeführt werden. Die bereits ausgewerteten Stützstellen fließen dabei mit ein und gehen nicht verloren.

3.1.2.2.K. Bestimmung der Abstiegsrichtung und der Schrittweite

Dieser Schritt ist Schritt C, dem steilsten Abstieg, sehr ähnlich. Der einzige Unterschied besteht darin, daß ein eventuell vorliegendes Minimum aus der kanonischen Analyse oder der Ridge Analysis bei der Bestimmung der Abstiegsrichtung mit berücksichtigt wird. Der Abstieg geschieht entlang des Vektors, der genau mittig zwischen der Richtung des steilsten Abstiegs und der Richtung des gefundenen Minimums liegt. Wurde bisher kein Minimum gefunden, dann entfällt auch dieser Unterschied und der Ablauf dieses Schrittes ist mit Schritt C identisch.

3.1.2.3. Terminierung

Dem traditionellen Ablauf der Response Surface Methode aus Abb. 3.2 folgend wird so lange iteriert, bis zum ersten Mal ein Polynom 2. Ordnung zum Einsatz kommt. Nachdem dieses ausgewertet und ein neuer Mittelpunkt des lokalen Bereichs bestimmt worden ist, wird das Verfahren beendet. Neddermeijer et al. lehnen dieses einfache Terminierungskriterium jedoch ab. Selbst für einfache Response Surfaces ist die Approximation durch Polynome 1. Ordnung oft unzureichend ist. Hierdurch wird die erste Phase der Response Surface Methode schnell verlassen und eine Approximation durch ein Polynom 2. Ordnung findet relativ früh statt. Damit ist es möglich, daß nach der ersten solchen Approximation noch lange nicht das gesuchte Optimum erreicht ist. Das Verfahren wird also zu fürh abgebrochen.

Ein von Neddermeijer et al. vorgeschlagenes alternatives Kriterium besteht darin, das Verfahren zu beenden, wenn der lokale Bereich zu klein wird. Wir notieren nach jeder linearen Suche, ob diese einen Fortschritt gebracht hat, oder die bisherige beste Faktorenbelegung weiterhin Bestand hat. Ergibt sich über zwei aufeinander folgende Suchen hinweg keine neue Faktorenbelegung, dann verringern wir den lokalen Bereich um einen zuvor festgelegten Faktor. Wir nehmen dabei an, daß das Verfahren mit der aktuellen Schrittgröße keine Fortschritte mehr erzielt, weil es sich in der Nähe eines lokalen Optimums befindet. Mit Hilfe des verkleinerten lokalen Bereichs können das Optimum genauer eingrenzen. Wird allerdings eine festgelegte Anzahl solcher Verkleinerungen erreicht, dann gilt das Optimum als ausreichend genau bestimmt und das Verfahren wird beendet.

3.1.2.4. Einbindung stochastischer Ranking & Selection-Verfahren

Peter Kissmann

Die wohl sinnvollste Stelle, um Ranking & Selection-Verfahren (siehe Kapitel 6) in die Response Surface Methode zu integrieren, bietet Schritt C, die Suche nach dem steilsten Abstieg. Da hier stets zwei Punkte im Suchraum miteinander verglichen werden sollen, aber die Zielfunktionen zumeist verrauscht sind, bietet es sich an, bei diesem Vergleich die beiden Punkte jeweils mehrfach auszuwerten, so daß das Rauschen weitestgehend unterdrückt wird und die Punkte vernünftig verglichen werden können, was durch die Verwendung der R&S-Verfahren realisiert werden kann.

3.1.3. Klassisches Experimental Design

Michael Schaten

3.1.3.1. Full factorial Design (2^k-Design)

Hierbei handelt es sich im Grunde um eine Ansammlung von Ansätzen, Methoden und Analyse-Techniken, die schon einige Jahrzehnte bekannt und gut dokumentiert sind. Das Hauptziel von *experimental Design* ist, abzuschätzen, wie sehr sich Änderungen des Inputs im Output (*response*) bemerkbar machen.
Für ein simples Beispiel für diese Technik stelle man sich vor, man könne für jeden Input nur zwischen zwei Werten bzw. Leveln wählen. Dabei ist es mehr oder weniger beliebig wie diese Werte gesetzt werden, sie sollten jedoch gegensätzlich und nicht unrealistisch in Bezug zu dem zu simulierenden System sein.
Bei k Inputs liefert dies 2^k verschiedene Kombinationen der Inputs, jede stellvertretend für eine andere Konfiguration des Experiments (2^k *factorial design*).
In folgendem anschaulichen Beispiel sind die oben genannten beiden möglichen Werte zum einen das + und zum anderen das -. Eine *Design-Matrix* ermöglicht es auch nach Abschluß des Experiments, schnell zu erkennen, welche Parameter-Konstellation zu einer gewissen Antwort R_i gehört.

Lauf i	**Faktor 1**	**Faktor 2**	**Faktor 3**	**Response**
1	-	-	-	R_1
2	+	-	-	R_2
3	-	+	-	R_3
4	+	+	-	R_4
5	-	-	+	R_5
6	+	-	+	R_6
7	-	+	+	R_7
8	+	+	+	R_8

Tabelle 3.2.: Design-Matrix für ein 2^3-Experiment

Die Ergebnisse eines solchen Experiments können in vielerlei Hinsicht interessant sein. Der so genannte *main effect* eines Faktors (bspw. Faktor 2 in obiger Tabelle) ist definiert als der durchschnittliche Unterschied in der Antwort beim Übergang von + zu -. Dieser kann berechnet werden, indem man die Werte (+ oder -) des Faktors mit der jeweiligen Antwort multipliziert und dann durch 2^{k-1} dividiert. In diesem Beispiel ergäbe dies für den Faktor 2 (für die anderen Faktoren gilt analoges):

$$[(R_3 + R_4 + R_7 + R_8)/4] - [(R_1 + R_2 + R_3 + R_6)/4],$$

vereinfacht:

$$(-R_1 - R_2 + R_3 + R_4 - R_5 - R_6 + R_7 + R_8)/4$$

Desweiteren könnte man feststellen wollen, ob der Effekt eines Faktors an einen anderen Faktor gebunden ist. Dies würde man dann *interaction* nennen.
Um dies zu berechnen, multipliziert man die Spalten der zu untersuchenden Faktoren Zeile für Zeile (gleiche Werte der Faktoren in dieser einen Zeile ergeben als Zeichen ein +, ungleiche ergeben ein -), fügt das Zeichen der jeweiligen Antwort hinzu, addiert und dividiert durch 2^{k-1}. Wieder obiges Beispiel vorausgesetzt ergäbe diese Untersuchung für die Faktoren 1 und 3 folgendes:

$$(+R1 - R2 + R3 - R4 - R5 + R6 - R7 + R8)/4$$

Besteht eine *interaction* zwischen zwei Variablen, so können deren *main effects* nicht isoliert betrachtet und interpretiert werden.

Dies führt dann auch sofort zu den Grenzen dieser Art von Design: Zu jedem Design dieser Art gehört ein spezielles lineares Regressions-Modell. In diesem Modell gibt es einerseits Terme, in denen die Variablen unabhängig von anderen Variablen vorkommen (linearer Anteil) und andererseits Terme mit Kreuz-Produkten. Diese Kreuz-Produkte zwischen zwei (oder auch mehreren) Variablen stehen für *interactions*. Sie erschweren die Auswertung der main effects erheblich, da sich Kreuz-Produkte natürlich stärker in der Antwort Y niederschlagen als lineare Terme.
Ein weiterer Nachteil bei der Arbeit mit *full factorial designs* tritt bereits bei der Nutzung von gemäßigt vielen Faktoren auf. Die Anzahl der Läufe des Experiments explodiert geradezu, denn sie steigt exponentiell in der Anzahl der vorhandenen Faktoren. Um dieses Problem zu umgehen, geht man zum so genannten *fractional-factorial design* (2^{k-p}-Design) über (Beispiel dazu im folgenden Abschnitt). Dabei werden nicht alle Faktor-Kombinationen betrachtet, sondern nur eine zuvor sorgfältig ausgewählte Menge. Der Nachteil dieser Methode ist jedoch, daß man eventuell vorkommende *interactions* nicht mehr sicher ausschließen kann. Natürlich wird die Wahrscheinlichkeit eine *interaction* festzustellen geringer, je kleiner man die Menge der zu testenden Kombinationen von Eingangs-Faktoren wählt.
Der letzte Nachteil, der jedoch nicht so schwer wiegt wie die oben genannten, ist die Eigenschaft, daß sämtliche Antworten der Experimente stochastisch sind. Dies bedeutet, daß alle zuvor erarbeiteten Ergebnisse, wie das feststellen von *interactions* oder *main effects* ebenfalls stochastischen Schwankungen in einem bestimmten Varianzbereich unterliegen. Jedoch besteht bei einer Simulation (im Gegensatz zu einem physikalischen Experiment) die Gelegenheit der *Replikation* (unabhängige Wiederholungen des Experiments mit gleichen Konfigurationen), wodurch diese Varianz reduziert werden kann. Ebenfalls möglich wäre die mehrfache Wiederholung nicht nur dieses einen Experiments, sondern des gesamten Designs. Dies hätte zur Folge, daß man ein sicheres Intervall, das so genannte *Konfidenzintervall*, für die erwarteten *main effects* und *interactions* angeben könnte.

3.1.3.2. fractional-factorial Design oder auch 2^{k-p}-Design

Die Technik, lediglich eine ausgewählte Menge (Fraktion) an Input-Faktoren zu benutzen, wird vor allem im Bereich von *screening Experimenten* genutzt. Dabei handelt es sich um Experimente, die zu Beginn eines Projektes durchgeführt werden, wodurch Variablen mit einem geringen Einfluß auf die Antwort von Variablen mit großem Einfluß unterschieden werden. Die als *wichtig* identifizierten Variablen werden dann im Weiteren intensiver untersucht. Der erfolgreiche Gebrauch von *fractional factorial Design* basiert auf drei zentralen Ideen:

- Die Seltenheit von Effekten: Existieren zahlreiche Variablen, so wird das System primär von einigen *main effects* oder (*low-order*)-interactions gesteuert (low-order-interactions sind interactions mit einem niedrigen Exponenten).
- Die Vorhersagbarkeit: Fractional factorial Designs können in stärkere (bzw. größere) Designs übertragen werden.
- Sequentielles Experimentieren: Es ist möglich die Durchläufe von zwei (oder auch mehreren) fractional factorial Experimenten zu kombinieren, um daraus ein größeres Design zu konstruieren.

Beispiel für ein one-half fraction eines 2^k Designs:
Angenommen sei eine Situation, in der drei Faktoren mit jeweils zwei Zuständen interessant sind, jedoch sind die Experimentierenden nicht dazu in der Lage alle $2^3 = 8$ Durchläufe laufen zu lassen, sondern lediglich die Hälfte aller möglichen Kombinationen, also 2^{3-1}=4. Da bei einer Reduzierung des Exponenten um 1 die Anzahl der Kombinationen halbiert wird, spricht man vom so genannten *one-half fraction Design*.
Tabelle 3.3 zeigt ein ähnliches Bild wie Tabelle 3.2, nur diesmal mit den Input-Faktoren A, B und C. Wir nehmen an, daß wir zur Halbierung der Durchläufe nur noch die Läufe a, b, c und abc weiter benutzen (diese stehen in der Tabelle in der oberen Hälfte der Tabelle).
a steht dabei für eine Faktor-Kombination, bei der lediglich der Faktor a mit dem Wert + belegt ist, b und c besitzen - als Wert. (Analog erklären sich die anderen Kombinations-Bezeichnungen in der linken Spalte in Tabelle 2). Die Kombination (1) bedeutet, daß alle Faktoren auf - gesetzt sind. Zu bemerken ist noch, daß nur Durchläufe für dieses $2^{3-1}-$Design genommen wurden, die in der ABC-Spalte ein + haben. ABC wird dann der *Generator* dieser bestimmten Fraktion genannt (manchmal auch einfach nur *Wort*). Des Weiteren gibt es auch in der Identitäts-Spalte nur +-Zeichen, daher wird

$$I = ABC$$

die definierende Relation für dieses Design sein. Im Allgemeinen kann noch erwähnt werden, daß die definierende Relation immer ein Set ist, das der Identität bzw. der Identitäts-Spalte gleicht. Main effects von A, B und C (Erklärung zum Erhalt dieser Formeln s.o.):

$$l_A = 1/2(a - b - c + abc)$$

Kombination	I	A	B	C	AB	AC	BC	ABC
a	+	+	-	-	-	-	+	+
b	+	-	+	-	-	+	-	+
c	+	-	-	+	+	-	-	+
abc	+	+	+	+	+	+	+	+
ab	+	+	+	-	+	-	-	-
ac	+	+	-	+	-	+	-	-
bc	+	-	+	+	-	-	+	-
(1)	+	-	-	-	+	+	+	-

Tabelle 3.3.: Variablen-Belegung für ein 2^3-factorial Design

$$l_B = 1/2(-a + b - c + abc)$$
$$l_C = 1/2(-a - b + c + abc)$$

interactions:

$$l_{BC} = 1/2(a - b - c + abc)$$
$$l_{AC} = 1/2(-a + b - c + abc)$$
$$l_{AB} = 1/2(-a - b + c + abc)$$

Zu erkennen ist, daß $l_A = l_{BC}$, $l_B = l_{AC}$ und $l_C = l_{AB}$ ist, es ist also unmöglich bspw. zwischen A und BC zu unterscheiden. Zwei oder mehr Effekte, die diese Eigenschaft besitzen, nennt man *alias*. In diesem Beispiel sind A und BC alias, B und AC alias und C und AB alias. Diese Beziehungen werden (für das Beispiel A und BC) durch $l_A \to A{+}BC$ signalisiert. Die alias-Beziehungen herauszufinden ist übrigens auch einfach über die definierende Relation möglich. Dabei multipliziert man alle Spalten mit der definierenden Relation:

$$A * I = A * ABC = A^2BC$$

(da das Quadrat einer Spalte gleich der Identität ist gilt:)

$$A = BC$$

(analoges gilt natürlich auch für B und C). Diese *half-fraction* mit $I = +ABC$ wird gewöhnlich als *principal fraction* bezeichnet, dazu gibt es auch die alternative *komplementäre one-half fraction*, bei der die Durchläufe (1), ab, ac und bc (zu sehen in Tabelle 3.3 in den letzten 4 Zeilen), benutzt werden. Das hat zur Folge, daß die definierende Relation hier nun

$$I = -ABC$$

lautet. In der Praxis ist es gleichgültig, welche dieser beiden Fraktionen gewählt wird, denn beide gehören der gleichen Familie an, und zusammen bilden sie ein vollständiges 2^3Design. Sei nun angenommen, daß nach dem Durchlauf der ersten Fraktion auch die zweite Fraktion von Durchläufen praktiziert wird; es sind nun also alle 8 möglichen Durchläufe getätigt worden. Nun hätten wir Abschätzungen, die frei sind von jedweden

alias-Beziehungen. Dies geschieht jedoch genauso gut durch das Addieren und Subtrahieren der Linear-Kombinationen der Effekte (main effects und interactions) der einzelnen Fraktionen. Betrachtet am Beispiel von A ergäbe dies:

$$l_A \rightarrow A + BC$$

$$l`_A \rightarrow A - BC$$

$$\Longrightarrow 1/2(l_A + l`_A) = 1/2(A + BC + A - BC) \rightarrow A$$

$$\Longrightarrow 1/2(l_A - l`_A) = 1/2(A + BC - A + BC) \rightarrow BC.$$

Für alle Linear-Kombinationen ergäbe dies dann:

i	**Von** $\frac{1}{2}(l_i + l`_i)$	**Von** $\frac{1}{2}(l_i - l`_i)$
A	A	BC
B	B	AC
C	C	AB

Tabelle 3.4.: Alle Alias-Beziehungen der drei Faktoren A, B und C

Design Resolution:
Das vorhergehende Beispiel war ein so genanntes Resolution III design. In diesem Design bekommt ein main effect als Alias eine Zwei-Faktor interaction. Ein Design ist von einer Resolution R, wenn kein p-Faktor Effekt einen Alias bekommt, der weniger als R-p Faktoren faßt. Für die Resolutions-Zahl wird im Allgemeinen eine römische Ziffer genutzt. Das zuvor gefertigte fractional factorial Design Modell wäre also ein 2^{3-1}_{III}.
Designs der Resolution III, IV und V sind im speziellen interessant und werden daher im Folgenden vorgestellt:

- Resolution III Designs
  In dieser Art des Designs bekommen main effects niemals einen anderen main effect als Alias, sondern nur Zwei-Faktor interactions. Interactions hingegen erhalten auch interactions als Alias. Ein Beispiel hierfür ist das vorangegangene mit I = ABC.
  Es ist also weder möglich main effects zweifellos zu identifizieren, noch interactions.

- Resolution IV Designs
  Dies sind Designs, bei denen kein main effect einen anderen main effect als Alias besitzt oder mit Zwei-Faktor interactions in Beziehung steht. Zwei-Faktor interactions erhalten jedoch andere Zwei-Faktor interactions als Alias. Ein 2^{4-1}-Design mit I = ABCD wäre ein Beispiel dafür.
  Mit diesem Design ist es also möglich, main effects eindeutig zu identifizieren, während jedoch interactions andere interactions als Alias haben.

- Resolution V Designs
  Dies sind Designs, bei denen kein main effect oder eine Zwei-Faktor interaction

einen anderen main effect oder eine andere Zwei-Faktor interaction als Alias besitzt. Zwei-Faktor interactions besitzen als Alias Drei-Faktor interactions. Ein Beispiel hierfür wäre ein 2^{5-1}-Design mit I = ABCDE.
Diese Resolution garantiert also eine Identifikation von sowohl main effects als auch interactions.

Im Allgemeinen ist zu sagen, daß die Resolution eines 2-leveligen fractional factorial Designs gleich der kleinsten Anzahl Buchstaben eines jeden Wortes der definierenden Relation ist.
Gewöhnlich benutzt man fractional Designs mit der höchstmöglichen Resolution, die mit dem Grad der Fraktionalisierung übereinkommt. Je höher die Resolution ist, desto einfacher ist es zu entscheiden, ob eine interaction vernachlässigbar ist oder nicht.
Das Vorgehen für die Konstruktion eines beliebigen one-half fraction Designs entspricht im Grunde dem im vorhergegangenen Beispiel:
Eine one-half fraction von einem 2^k-Design mit einer höchsten Resolution wird konstruiert, indem man zuerst ein sog. *basic design* erstellt. Dieses besteht aus einem full 2^{k-1}-factorial Design, an den dann die k-te Spalte als Linearkombination (als interaction) der vorherigen k-1 Spalten angehangen wird.

3.2. Programmbeschreibung

3.2.1. Anforderungsbeschreibung

Thomas Hutter, Jan Kriege

Im weiteren Verlauf wird nur auf die Anforderungen an die RSM-Gruppe eingegangen. Hier galt es, die Response Surface Methode gemäß der Ablauffestlegung aus dem Paper von Neddermeijer et al. in ein Programm umzusetzen. Zum genauen Aufbau dieser Festlegung sei an dieser Stelle auf das entsprechende Kapitel verwiesen. Zusätzlich zu dem in der Literatur beschriebenen Ablauf für Parameter mit kontinuierlichem Wertebereich sollte die Response Surface Methode auch in der Lage sein, Modelle mit Faktoren zu optimieren, die einen diskreten Wertebereich haben, was einige Änderungen am Ablauf des Programms erfordert. So müssen zum Beispiel die ermittelten Designpunkte auf diskrete Werte gerundet werden. Zur Vereinfachung wird angenommen, daß alle diskreten Faktoren eine Schrittweite von eins haben. Andere Schrittweiten lassen sich in der GUI über die Umrechnung von Faktor- und Modellvariablen realisieren. Eine ausführlichere Darstellung der notwendigen Änderungen erfolgt im Rahmen der Klassenbeschreibung. Das Programm sollte sowohl unter Windows als auch unter Unix und Linux lauffähig sein.

Zudem erfolgte eine Anbindung an die Whitebox-Klasse der GUI, die einerseits die für RSM benötigten Parameter liefert und andererseits eine Schnittstelle zu den Benchmarkfunktionen und den Modellierungstools APNN-Toolbox und ProC/B darstellt.

Vor allem im Bereich der mehrdimensionalen Optimierung wurde versucht, eine gute Belegung von Eingabeparametern zu erzielen. Einzelne Parameter des Algorithmus' las-

sen sich mit Hilfe der GUI auf beliebige Werte setzen, um eine Anpassung des Verfahrens an das vorliegende Problem zu ermöglichen.

Ein wesentliches Problem bei der Optimierung von Simulationsmodellen ist, daß die Algorithmen mit den mehr oder weniger starken stochastischen Schwankungen der Simulationsausgabe zurechtkommen müssen. Der Algorithmus sollte einen voll automatisierten Ablauf darstellen, um unter anderem auf diese Art von Problemen zu reagieren. Dieses war auch die größte Schwierigkeit während der Programmierung. Denn vor diesem Versuch in der PG existierte ein solcher Ansatz noch nie, so daß wir auf keinerlei Kenntnisse und Forschungsergebnisse zurückgreifen konnten. Da im Laufe der Optimierung mehrere Entscheidungen zu treffen sind, mußten wir den Ablauf auf eine Anzahl bestimmter Verfahren beschränken, um eine manuelle Selektion zu vermeiden. Desweiteren kam hinzu, daß unterschiedliche Ergebnisse während der Laufzeit auch zu unterschiedlichen Entscheidungen führen. So kann es sein, daß aufgrund einer schlechten Auswahl von Kriterien die gesamte Optimierung ohne ein gefundenes Minimum terminiert, oder verfälschte Ergebnisse liefert.
Die Gruppe versuchte dieses Problem mit Hilfe der Selbstkorrektur in den Griff zu bekommen, welche es dem Algorithmus ermöglicht, selbstständig in den Ablauf einzugreifen und Entscheidungen zu treffen.
Er entscheidet ohne äußeren Einfluß, welche Verbesserungen durchgeführt werden müssen, um die Approximation zu verbessern und welches Maß an Abweichung noch akzeptabel ist. Bei der Güte von einigen Zwischenergebnissen haben wir uns auf eine obere Schranke festgelegt, um dem Programm einige Entscheidungen zu erleichtern.
Inwieweit diese vorgegebenen Entscheidungen sinnvoll gewählt wurden, ist in den Leistungsstudien genauer dokumentiert.
Am Ende einer komplett durchgeführten Optimierung werden die gefundenen Ergebnisse an die GUI weitergereicht und dort für den Benutzer dargestellt.

3.2.2. Programmaufbau

Bartosz Fabianowski, Jan Kriege

Der RSM Optimierer ist objektorientiert aufgebaut, besitzt jedoch absichtlich eine einfache und flache Klassenhierarchie. Auf diese Weise existieren nur wenige schmale Schnittstellen, was der verteilten Programmierung in einer Gruppe sehr zuträglich ist. Das gesamte Programm implementiert weitestgehend den Ablauf der Response Surface Methode nach dem Paper von Neddermeijer et al. Dort, wo das Programm Entscheidungen treffen muß, haben wir jedoch die im Paper fehlenden konkreten Heuristiken ergänzt. Außerdem gibt es einige Abweichungen vom Ablauf, falls Parameter mit diskretem Wertebereich verwendet werden.
Die Steuerung des gesamten Ablaufs geschieht zentral durch die Klasse *Ablaufsteuerung*. Sie durchläuft in einer Schleife die in Abb. 3.3 dargestellten 11 Schritte, bis eines der Terminierungskriterien erfüllt wird. Die *Ablaufsteuerung* ist auch die einzige Klasse, die

mit der außerhalb der RSM Gruppe entwickelten *BlackBox* kommuniziert. Sie liest bei Programmstart mit Hilfe der *BlackBox* die in der GUI angegebene Konfiguration ein und stellt die ermittelten Parameterwerte den anderen Klassen zur Verfügung.

Zur Übergabe der Parameter dient eine Instanz der Klasse *Parameter*. Die einzige Aufgabe dieser Klasse ist es, als zentrales Datenreservoir zu dienen. Die übrigen Klassen enthalten nur statische Methoden und haben keine eigene Datenhaltung. Sämtliche während der Optimierung benötigten oder errechneten Werte werden über die Klasse *Parameter* ausgetauscht. Ein Vorteil dieses Ansatzes besteht in seiner großen Transparenz. Da jeglicher Datenaustausch über die Klasse *Parameter* stattfindet, können alle sich während der Optimierung ändernden Werte zentral eingesehen werden. Dies ist insbesondere beim Debugging wertvoll, weil keine privat übergebenen Parameter das Verhalten einer Klasse auf nicht nachvollziehbare Weise beeinflussen können.
Nachdem eine Instanz von *Parameter* angelegt und mit den von der GUI übergebenen Parameterwerten gefüllt worden ist, beginnt die *Ablaufsteuerung* ihre Hauptschleife. Dabei führt sie selbst keinerlei Berechnungen durch, sondern delegiert diese an die sechs Klassen, die jeweils einen inhaltlich zusammenhängenden Teil der Algorithmen und Heuristiken implementieren. Bei der Response Surface Methode wird die Response Surface des zu optimierenden Modells durch Polynome 1. und 2. Ordnung approximiert. Zu den meisten Schritten, die auf Polynomen 1. Ordnung operieren, existieren Pendants, die weitgehend identische Berechnungen für Polynome 2. Ordnung durchführen. In solchen Fällen sind die analogen Berechnungsmethoden für 1. und 2. Ordnung immer in derselben Klasse zusammengefaßt, um den inhaltlichen Zusammenhang zu wahren.
Die genauen Aufgaben der einzelnen Klassen können ihren Beschreibungen im entsprechenden Kapitel dieses Berichtes entnommen werden. An dieser Stelle sind nur ihre groben Aufgabenbereiche angegeben. *ED* bestimmt ein Experimental Design und veranlaßt Auswertungen des zu optimierenden Modells an den Stützstellen. Bei Faktoren mit diskretem Wertebereich nutzt *ED* zusätzlich einige Methoden der Klasse *RSMdiskret*, die Hilfsmethoden für RSM mit diskreten Parametern zur Verfügung stellt, zum Runden und zur Überprüfung der Stützstellen. *OLS* erstellt anschließend ein Approximationspolynom und *anova* überprüft, ob die Approximation ausreichend gut ist. Wenn nicht, versucht *Korrektur*, durch Änderungen an den Parametern des Optimierers die Voraussetzungen für eine bessere Approximation zu schaffen. Sobald ein akzeptables Approximationspolynom vorliegt, sucht *LineareSuche* in der Richtung des steilsten Abstiegs nach der besten Parameterbelegung. Zur Bestimmung dieser Parameterbelegung nutzt die *LineareSuche* Ranking & Selection (siehe 3.1.2.4). Auch hier werden bei Faktoren mit diskretem Wertebereich wieder Methoden aus *RSMdiskret* verwendet. Bei Approximation durch ein Polynom 2. Ordnung werden zur Bestimmung der Abstiegsrichtung zusätzlich die Ergebnisse der in *Analysen* implementierten kanonischen und Ridge Analyse genutzt.
Zusätzlich zu den bisher beschriebenen Klassen existiert noch *RSMEA*. Ähnlich der Klasse *Parameter* in ihrer Rolle als zentrale Datenhaltung dient sie als zentraler Anlaufpunkt für Ausgaben. *RSMEA* sammelt alle Ausgaben und gibt diese an die GUI weiter, wo sie für den Nutzer ausgegeben werden. Zusammen mit der *BlackBox* stellt *RSMEA*

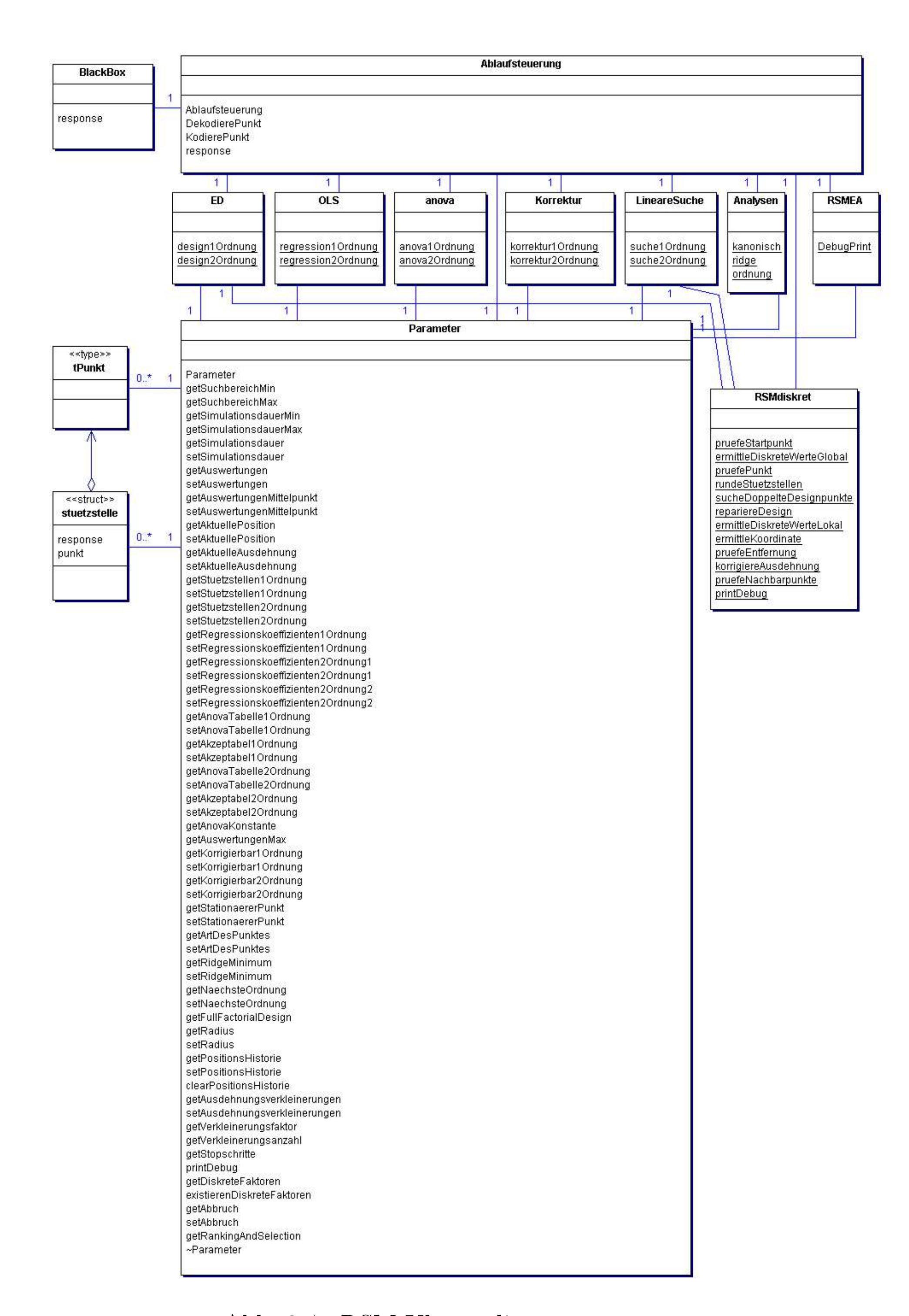


Abb. 3.4.: RSM Klassendiagramm

die einzige Schnittstelle von RSM zu den anderen Programmteilen dar.

3.3. Klassenbeschreibung

Thomas Hutter, Normann Powierski, Jan Kriege

3.3.1. Ablaufsteuerung

Die *Ablaufsteuerung* ist die zentrale Klasse des RSM Algorithmus, die den gesamten Ablauf vom Aufruf bis zur Ausgabe steuert. Der Algorithmus wird gestartet indem die *Whitebox* eine Instanz der Ablaufsteuerung erzeugt. Die einzigen Parameter, die dabei übergeben werden, sind eine Referenz auf die *Blackbox*, die alle benötigten Parameter zur Verfügung stellt, und eine Variable, in der das gefundene Optimum gespeichert wird. Der gesamte Ablauf findet dann innerhalb des Konstruktors statt.

Zuerst werden mit Hilfe der Blackbox die Parameter ausgelesen, die der Nutzer in der GUI angegeben hat. Dies geschieht von der Ablaufsteuerung und nicht indem die Blackbox die benötigten Parameter übergibt, um die Auswahl der benötigten Parameter flexibel zu halten. Falls das Auslesen der Parameter erfolgreich war, wird eine Instanz der Klasse *Parameter* erzeugt, wobei die entsprechenden Variablen übergeben werden.
Nun beginnt der eigentliche RSM-Ablauf nach Neddermeijer et al.; zu den Details sei hier auf den Abschnitt 3.1.1 „Response-Surface-Methode - Was ist das“ verwiesen. Für den Fall, daß einige der Faktoren einen diskreten Wertebereich haben, kommt es zu Abweichungen von diesem Ablauf: Zu Beginn wird überprüft, ob die Startposition erlaubt ist und die Koordinaten des Punktes falls nötig auf diskrete Werte gerundet. Außerdem wird bei diskreten Faktoren die Ridge-Analyse nicht durchgeführt, da diese nicht mehr zuverlässig funktionierte.
Die Klasse Ablaufsteuerung verändert (im Konstruktor) selbst keine Variablen der Parameterklasse, sondern liest diese nur aus, um Entscheidungen über den weiteren Ablauf treffen zu können.
Der Ablauf endet, wenn eins der Abbruchkriterien erreicht wurde. Da sich die RSM-Gruppe dazu entschlossen hat bei Polynomen 2. Ordnung immer eine Korrektur zu versuchen (siehe Klasse *Korrektur* in Abschnitt 3.3.5), ist das einzige Abbruchkriterium innerhalb der Methode `fortschritt` implementiert. Die Methode `fortschritt` prüft an mehreren Stellen des Ablaufs, ob innerhalb einiger Schritte eine Veränderung der aktuellen Position stattgefunden hat. Wenn kein Fortschritt erkennbar ist, wird eine Verkleinerung des lokalen Bereichs um einen in der GUI einstellbaren Faktor vorgenommen. Die Anzahl der erfolgten Verkleinerungen wird mitgezählt. Nach einer ebenfalls in der GUI einstellbaren Anzahl meldet die Klasse `fortschritt` Stillstand und der Algorithmus endet. Bei diskreten Faktoren erfolgt nach dem Abbruch noch eine Sonderbehandlung, bei der die Nachbarpunkte der aktuellen Position untersucht werden, da sich im Rahmen der Leistungsstudien herausstellte, daß dieses Vorgehen das gefundene Optimum oft noch verbessert.
Simulationsaufrufe der einzelnen Klassen werden über die Methode `response` realisiert.

Diese erwartet einen lokalen Punkt als Übergabe, rechnet diesen in globale Koordinaten um und ruft damit dann die Evaluate Methode der *Blackbox* auf. Innerhalb der `response` Methode ist ein Zähler realisiert, welcher eine Auswertung der benötigten Simulationsaufrufe ermöglicht.
Ebenfalls Aufgabe der *Ablaufsteuerung* ist das Umrechnen von Punkten aus dem globalen Zahlenbereich, der von der Blackbox verwendet wird, in den intern verwendeten Bereich von -1 bis 1 und umgekehrt. Dies geschieht in den Methoden `kodierePunkt` und `dekodierePunkt`

3.3.2. ED

Die Klasse *ED* bietet je eine Funktion zum Erstellen des Designs 1. und 2. Ordnung an. Beim Design 1. Ordnung wird zunächst aus der Parameterklasse ausgelesen, ob ein Full Factortial Design oder ein Fractional Factorial Design erstellt werden soll. Für das Full Factorial Design werden einfach alle Kombinationen aus $+1$ und -1 Werten für die Faktoren erstellt. Das Fractional Factorial Design wurde so implementiert, daß zunächst ein geeignetes p bestimmt wird und danach die Generatoren erzeugt werden.
Für das Design 2. Ordnung wird das Design aus dem ersten Schritt zunächst so erweitert, daß es mindestens Resolution V hat. Als nächstes wird versucht, ein Central Composite Design zu erstellen. Sollte dabei der erlaubte Suchbereich verlassen werden, wird statt dessen ein Face Centered Design erzeugt.
Als letzter Schritt werden sowohl für das Design 1. Ordnung als auch für das Design 2. Ordnung die Stützstellen ausgewertet.
Falls einige Faktoren einen diskreten Wertebereich haben, sind auch hier wieder einige Sonderfälle zu beachten. Zunächst werden die Koordinaten mit diskretem Wertebereich für alle Designpunkte gerundet. Da es bei einem kleinen Suchbereich passieren kann, daß durch das Runden Designpunkte doppelt vorkommen, wird danach überprüft, ob eine Korrektur des Designs nötig ist. Dabei wird versucht, für die doppelten Punkte andere Koordinaten zu finden. Falls der Suchbereich so klein ist, daß keine sinnvollen Koordinaten mehr gefunden werden können, wird die Ausdehnung für den betreffenden Faktor auf 0 gesetzt. Dies hat zur Folge, daß der aktuelle Wert des Faktors festgehalten wird, um mit den restlichen Faktoren weiter optimieren zu können. Für diese Operationen nutzt *ED* Funktionen, die die Klasse *RSMdiskret* zur Verfügung stellt.

3.3.3. OLS

Die Klasse `Ordinary Least Square` kurz *OLS* dient zur Berechnung der Regressionskoeffizienten, um eine geeignete Approximation durch ein Polynom erster oder zweiter Ordnung durchzuführen. Sie beinhaltet eine Methode für jede Ordnung.

Beim Aufruf der Methode für die erste Ordnung, wird mit Hilfe eines Zeigers auf die Klasse Parameter zugegriffen, aus der die benötigten Vektoren mit den Stützstellen und den zugehörigen Responsewerten geholt werden. Anschließend finden die einzelnen Berechnungen von *OLS* statt. Zu Beginn wird eine Matrix mit allen vorhandenen Stützstellen aufgebaut, wobei die erste Spalte dieser Matrix immer mit Einsen belegt ist.

Anschließend muß diese Matrix transponiert und mit ihrer Inversen multipliziert werden. Das entstehende Produkt muß zuletzt noch invertiert und mit dem Vektor der Responsewerte multipliziert werden. Da diese Berechnungen sehr kompliziert werden können, wurde hier auf die `GNU Scientific Library` kurz `GSL` zurückgegriffen. Unter Zuhilfenahme der `GSL` konnte eine effiziente Berechnung dieser Gleichung erfolgen. Am Ende erhalten wir unseren gewünschten Vektor mit den Regressionskoeffizienten, welchen wir wiederum mit Hilfe eines Zeigers in die Klasse Parameter speichern.

Der Unterschied zur zweiten Ordnung besteht hauptsächlich im Aufbau der Matrix. Die anschließende Berechnung ist völlig analog zu der Methode für die erste Ordnung. Da es sich hier um eine Approximation zweiter Ordnung handelt, erhalten wir am Ende nicht nur einen Vektor, sondern auch eine Matrix mit den passenden Regressionskoeffizienten.

3.3.4. anova

Es stellt sich die Frage, ob das im vorhergehenden Abschnitt berechnete Polynom die Response Surface Funktion in angemessener Weise approximiert. Diese Entscheidung ist wichtig, denn sie ist ausschlaggebend für die Bestimmung des steilsten Abstiegs in der Klasse Lineare Suche. Wird die Funktion durch das Polynom nur unzureichend angenähert, dann kann die völlig falsche Richtung eingeschlagen werden.

Um eine solche Aussage über ein Polynom treffen zu können, wird zuvor eine Teststatistik berechnet. Dazu werden sowohl die Regressionskoeffizienten als auch die Stützstellen aus der Klasse Parameter benötigt. Die geeignete Teststatistik, für die wir uns entschieden haben, ermittelt den Quotienten der mittleren quadrierten Abweichung des Approximationspolynoms und der mittleren quadrierten echten Fehler. Ein großer Wert bedeutet, daß die Abweichungen durch die Approximation im Verhältnis zu den stochastischen Schwankungen der Response Surface Funktion nicht vernachlässigbar sind. Folglich handelt es sich um eine schlechte Approximation. Der konkrete betrachtete Wert p ergibt sich durch Einsetzen des Quotienten in die Verteilungsfunktion der F-Verteilung. Da diese als Quotient zweier χ^2-Verteilungen definiert und ihre Berechnung sehr aufwendig ist, haben wir auch hier auf die `GSL` zurückgegriffen und die Verteilung komplett eingebunden. Das Hauptproblem bei der Implementierung dieser Klasse besteht darin, einen geeigneten kritischen Wert für den Vergleich, sowohl bei erster als auch bei zweiter Ordnung, zu finden. Die von uns verwendete Konstante hat den Wert $0,09$.
Anhand dieser Vorberechnungen kann nun über die Güte eines Approximationspolynoms entschieden werden. Wird diese als akzeptabel erachtet, wird der Wert `true` andernfalls `false` in der Klasse *Parameter* gespeichert. Die Klasse *Ablaufsteuerung* reagiert dann auf diese Werte und entscheidet über den weiteren Verlauf des Algorithmus. Doch für eine genauere Erläuterung dieses Vorgehens sei auf die Klasse *Ablaufsteuerung* in Abschnitt 3.3.1 verwiesen.

3.3.5. Korrektur

Die Klasse *Korrektur* wird im Ablauf des Algorithmus' nur aufgerufen, wenn die Approximation eines Polynoms, egal ob erster oder zweiter Ordnung, nicht akzeptabel erscheint.

Es wird versucht, die Parameter des Optimierers so zu verändern, daß eine bessere Approximation gefunden werden kann, vorausgesetzt es besteht eine Aussicht auf Erfolg. Andernfalls wird an die Klasse *Parameter* eine Unkorrigierbarkeit gemeldet. Bei Polynomen erster Ordnung wird Korrigierbarkeit angenommen, wenn der von *anova* ermittelte p-Wert unter einer vorgegebenen Konstante liegt. Diese wird um den Faktor 4 größer angesetzt, als in der Klasse *anova*. Bei Polynomen zweiter Ordnung wird immer eine Korrektur der Parameter versucht, unabhängig vom p-Wert.

Fällt die Entscheidung, daß eine Korrektur vorgenommen werden soll, dann muß festgelegt werden, welche Parameter in welchem Ausmaß zu verändern sind. Hier gibt es unterschiedliche Möglichkeiten der Implementierung. Die RSM-Gruppe hat sich dafür entschieden, die Anzahl der Auswertungen pro Stützstelle zu variieren. Dazu wird diese so oft erhöht, bis sie ein gewisses Maximum erreicht. Dieses Maximum wird dabei über die GUI festgelegt. Sollte die Anzahl der Auswertungen den Maximalwert erreichen und immer noch keine akzeptable Approximation ermittelt werden können, wird die Korrektur abgebrochen und Unkorrigierbarkeit gemeldet.

3.3.6. LineareSuche

Die Klasse *LineareSuche* berechnet zum einen die Richtung, in die während der Optimierung gelaufen werden soll und zum anderen die Schrittweite des lokalen Bereichs. Um die Richtung berechnen zu können, wird mit Hilfe von Zeigern auf die Klasse *Parameter* zugegriffen und die Methode `getRegressionskoeffizienten` aufgerufen. Da die *LineareSuche* auf globale Variablen angewiesen ist, die Regressionskoeffizienten aber in lokaler Art vorliegen, muß zunächst eine Transformation erfolgen. Anschließend wird, da unser Programm minimieren soll, das Vorzeichen geändert, woraus sich dann unsere Richtung ergibt.

Die zweite Aufgabe dieser Klasse liegt darin, die Schrittweite des lokalen Bereichs zu bestimmen. Hierzu benötigt sie die beiden Parameter `AktuelleAusdehnung` und `AktuellePosition`, welche wiederum aus der Klasse *Parameter* ausgelesen werden. Zunächst initiiert die *LineareSuche* einen Simulatoraufruf über die *Ablaufsteuerung* und übergibt ihr die aktuelle Position. Wenn ein vorübergehend bester Responsewert gefunden wurde, wird dieser als aktueller Responsewert betrachtet. Auch hier ergeben sich aufgrund stochastischer Schwankungen teilweise große Diskrepanzen bezüglich der Responsewerte. Um diesem Effekt entgegen zu wirken, wurde zum einen ein Stopkriterium hinzugefügt, welches dem Programm erlaubt, auch mal einen schlechteren Responsewert zu akzeptieren. Es könnte sein, daß wir aufgrund von zu starkem Rauschen nur einen kleinen Anstieg überwinden müssen, um an ein besseres Optimum zu gelangen. Die Anzahl des Akzeptierens einer falschen Lösung kann auch über die GUI eingestellt werden. Zum anderen wurden an dieser Stelle Ranking & Selection-Verfahren eingebunden. So können die einzelnen Punkte mehrfach ausgewertet und damit das Rauschen weitestgehend unterdrückt werden. Auch diese Parameter lassen sich in der GUI einstellen. Nachdem wir den aktuellen Responsewert bestimmt haben, wird mit Hilfe der Schrittweite und der Ausdehnung die aktuelle Position ermittelt und in die Klasse *Parameter* gespeichert.

Das Verfahren für die *Lineare Suche* zweiter Ordnung ist im Prinzip identisch. Zumindest was die Bestimmung der Abstiegsrichtung angeht. Bei der Bestimmung der Richtung wird allerdings noch das Ergebnis der kanonischen Analyse mit berücksichtigt. Je nachdem, um welche Art von Optimum es sich dort handelt, reagiert die *LineareSuche* anders. Jedoch sei hier auf die Beschreibung der kanonischen Analyse in Abschnitt 3.3.7 verwiesen.
Für Faktoren mit diskretem Wertebereich ergeben sich einige Änderungen im Ablauf: Zunächst wird die gefundene Position nach jedem Schritt gerundet. Für kleine Schrittweiten kann dies allerdings zur Folge haben, daß sich die Koordinaten der diskreten Faktoren nicht ändern, da die Schrittweite nicht groß genug ist, um zu dem nächsten diskreten Wert zu gelangen. Deshalb wird nach jedem Schritt geprüft, ob sich die Werte der Faktoren geändert haben. Falls mehrmals hintereinander keine Änderung festgestellt wurde, wird die Position des Faktors einfach auf den nächsten diskreten Wert gesetzt.

3.3.7. Analysen

Die Klasse *Analysen* ist nur im Bereich der Approximation eines Polynoms der zweiten Ordnung relevant. Sie beinhaltet drei Methoden. Die kanonische Analyse, die Ridge Analyse und die Ordnung der nächsten Approximation.

Bei der kanonischen Analyse werden die Art und die Lage des stationären Punktes des approximierten Polynoms zweiter Ordnung bestimmt. Die Berechnung geschieht durch eine partielle Matrixinversion, die mit Hilfe von Matrixroutinen aus der `GSL` gelöst wird. Um die Effizienz dieser Berechnung zu steigern, wird die Matrix zuvor in eine symmetrische Form gebracht. Handelt es sich bei dem stationären Punkt um ein Minimum, dann ist es im Rahmen der Response Surface Methode interessant, die Response Surface Funktion in Richtung dieses Minimums zu verfolgen.

Wurde jedoch ein Maximum oder ein Sattelpunkt ermittelt, dann wird mit der Ridge Analysis fortgefahren. Diese führt eine Nebenbedingung ein und berechnet das absolute Minimum des Approximationspolynoms auf der resultierenden beschränkten Menge. So wird die Bestimmung eines Minimums, wenn auch nur unter einer willkürlichen Nebenbedingung, erzwungen. Dieses wird dann wiederum mit Hilfe der partiellen Matrixinversion berechnet.

Zuletzt wird noch die Ordnung der nächsten Approximation bestimmt. Dies geschieht zur Erkundung des gesamten Suchbereichs mit Hilfe eines Polynoms erster Ordnung oder durch eine genauere Eingrenzung des gefundenen Optimums durch ein Polynom zweiter Ordnung. Die Umschaltung zwischen beiden Polynomen sollte halbwegs intelligent erfolgen. Mangels einer besseren Heuristik entscheidet sich momentan der Algorithmus immer für die Approximation durch ein Polynom erster Ordnung.

3.3.8. RSMEA

Die Klasse *RSMEA* ist für alle Ausgaben zuständig. Hier laufen die Debugausgaben aller Klassen sowie die Ausgabe des Endergebnisses zusammen. Durch diese zentrale Schnittstelle wird es ermöglicht, unkompliziert alle Ausgaben an die GUI weiterzureichen.

3.3.9. RSMdiskret

Die Klasse *RSMdiskret* ist in dem RSM-Ablauf nach Neddermeijer et al. eigentlich nicht vorgesehen und stellt lediglich einige Hilsmethoden zur Verfügung, die von *ED*, *Ablaufsteuerung* und *LineareSuche* zur Behandlung von Faktoren mit diskretem Wertebereich genutzt werden.
Unter anderem enthält die Klasse Methoden zum Runden von Punkten, zur Suche und Korrektur von doppelten Designpunkten und zur Untersuchung von Nachbarpunkten der aktuellen Position.

3.3.10. Parameter

Die Klasse *Parameter* ist ein Kernbestandteil der flachen Architektur des RSM-Algorithmus. Hier werden alle Variablen, auf die mehrere Klassen zugreifen müssen, gespeichert und mittels Get- und Set-Methoden bereitgestellt. Durch diese zentrale Parameterklasse, die als Referenz jeder Klasse beim Aufruf übergeben wird, bleiben die Schnittstellen zwischen den einzelnen Klassen einfach und das Hinzufügen weiterer Variablen ohne die Gefahr von Schnittstellenproblemen möglich.
Es existiert nur ein einziger Konstruktor, der alle verwendeten Variablen als Übergabe erwartet, somit werden bei Aufruf mit gültigen Werten alle Variablen richtig vorbelegt.

3.4. Leistungsstudien

Marcus van Elst, Jan Kriege, (Christian Bäcker)

3.4.1. Anzahl Replikationen bei verrauschten Funktionen

Es sollte untersucht werden, welchen Einfluß eine Veränderung der Anzahl der Auswertungen bei verrauschten Funktionen auf das Ergebnis der RSM Methode hat. Dazu wurden vor allem die Schaffers, Ackley und Rastrigins Funktion untersucht.

Der Suchbereich wurde standardmäßig auf -2 bis $+2$ beschränkt. Die Ausdehnung betrug 1 und die Startposition wurde ebenfalls auf 1 gesetzt. Es wurden die Dimensionen 1, 2, 5 und 10 jeweils mit absolutem Rauschen bis ca. 20% untersucht. Die Anzahl der Auswertungen wurde von 1-10 hoch gesetzt.

Der beste gefundene Responsewert wurde bei steigender Varianz schlechter, überraschenderweise aber bei sehr starker Varianz (ca. 20%) wieder besser (Eine Begründung dieses Phänomens kann an dieser Stelle noch nicht gegeben werden). Es stellte sich heraus, daß bis zur 5. Dimension der beste gefundene Punkt, durch Erhöhung der Anzahl der Auswertungen (bis maximal 10) noch weiter verbessert werden kann. Bei höheren Dimensionen konnte durch die Erhöhung der Anzahl der Auswertungen pro Stützstelle kein verbessertes Ergebnis festgestellt werden, da bereits schon durch die hohen Dimensionen, eine erhebliche Anzahl an Auswertungen benötigt werden um das Full Factorial Design zu erstellen.

3.4.2. Full Factorial Design vs. Fractional Factorial Design

Es sollte untersucht werden, ob die Auswahl des Designs, also Full Factorial Design bzw. Fractional Factorial Design, zu einem unterschiedlichen Ergebnis führen. Beispielhaft wurde dazu die Sphere Funktion untersucht. Der Suchbereich wurde standardmäßig auf -2 bis $+2$ beschränkt. Die Ausdehnung betrug 1, die Startposition wurde ebenfalls auf 1 gesetzt und die Anzahl der Auswertungen pro Stützstelle betrug 2. Als Rauschen wurde absolute Varianz zwischen 0 und 20% verwendet und die Dimensionen wurden schrittweise von 1 bis 10 erhöht.

Insgesamt können zwei Aussagen getroffen werden. Bei wenig Rauschen ist das Fractional Factorial Design dem Full Factorial Design klar überlegen, da die Anzahl der Auswertungen beim Fractional Factorial Design deutlich geringer ist, die Ergebnisse aber etwa gleich gut sind.

Bei stärkerem Rauschen und höheren Dimensionen, ist das Full Factorial Design allerdings erheblich besser, da sich dann die Vorteile des komplexeren Designs bemerkbar machen, und so der größere Aufwand für die Erstellung gerechtfertigt ist.

3.4.3. Startposition, Lokaler Bereich

Es sollte überprüft werden, wie sich die Startposition, sowie die Größe des lokalen Suchbereiches auf das Ergebnis auswirkt. Des Weiteren sollte überprüft werden, ob Veränderungen des lokalen Bereiches zu unterschiedlichen Ergebnissen führen.

Es wurden alle gegebenen Benchmarkfunktionen in allen Dimensionen bis 10 getestet. Der Suchbereich wurde standardmäßig auf -2 bis $+2$ beschränkt. Die Ausdehnung betrug 1, die Startposition wurde anfänglich ebenfalls auf 1 gesetzt und die Anzahl der Auswertungen pro Stützstelle betrug 2. Als Rauschen wurde relative Varianz zwischen 0 und 50% verwendet. Als erstes Ergebnis läßt sich sagen, daß die Einstellungen sehr stark von den verwendeten Funktionen und deren Rauschen abhängen. Es lies sich also keine global optimale Konfiguration finden. Allgemein läßt sich sagen, daß ein großer lokaler Suchbereich gut ist, um lokale Minima zu überspringen, und um größere Entfernungen zum Optimum möglichst schnell zu überbrücken. Insgesamt sind die Ergebnisse allesamt leider sehr grob und die standardmäßig verwendeten Einstellungen lagen in der „goldenen Mitte“.

Eine Untersuchung des Faktors zur Verkleinerung des lokalen Bereichs wird an der Rastrigins Funktion untersucht. Bei einer Dimension von 1 und einem Verkleinerungsfaktor von 2 ergeben sich bei geringem Rauschen schlechte Werte. Bei großem Verkleinerungsfaktor verfängt der Algorithmus sich schnell im lokalen Minimum. Fazit: Bei sehr hoher und sehr kleiner Dimension liefert Faktor 5 beste Werte.

3.4.4. Einfluß des p-Wertes in der Anova Klasse

Es sollte der Einfluß des p-Wertes für den Anova Algorithmus untersucht werden. Dazu wurden die Ackley, Schaffer und Sphere Funktion in den Dimensionen 2, 5 und 10 mit relativem Rauschen von 0 bis 50% in 5% Schritten untersucht. Der Suchbereich wurde standardmäßig auf -2 bis $+2$ beschränkt. Die Ausdehnung betrug 1 und die Startposition

wurde ebenfalls auf 1 gesetzt. Der zu untersuchende Anova Wert (p) wurde von $0,05$ bis $0,2$ in 0,01er Schritten erhöht. Um eine präzise Aussage treffen zu können, wurden von jedem RSM Durchlauf 5 Replikationen erstellt.

Als Ergebnis kann ganz klar gesagt werden, daß sich ein p-Wert von $0,09$ als optimal herausgestellt hat. Dies ist damit zu begründen, daß ein approximiertes Polynom, welches von OLS erstellt wird, in erster Linie dazu dient, eine Suchrichtung anzugeben. Solange also die Suchrichtung stimmt, muß das Polynom nicht besonders gut an die tatsächliche Funktion angenähert werden. Der optimale Wert ergibt sich nun daraus, daß bei einem kleinen p-Wert z.B. $0,05$ das Polynom oftmals als zu schlecht erkannt wird, und der RSM Algorithmus einige Korrekturmöglichkeiten starten muß, die einen deutlichen Mehraufwand an Auswertungen bedeuten. Bei dem als optimal gefundenen p-Wert $0,09$ kann der Algorithmus oftmals mit dem approximierten Polynom weiterlaufen und kommt somit schneller zu einem möglichen Optimum.

3.4.5. Verkleinerungsfaktor für den lokalen Bereich

Es sollte untersucht werden, wie sich die Veränderung der Suchbereichsgröße auf den Algorithmus auswirkt. Dazu konnte einmal der Verkleinerungsfaktor und die Anzahl der Verkleinerungen variiert werden. Es wurden alle Benchmarkfunktionen mit absolutem Rauschen von 0% bis 20% in den Dimensionen 1 bis 10 untersucht. Der Verkleinerungsfaktor wurde bei 2, 5, 10, 20 und 50 untersucht. Die Anzahl der Verkleinerungen wurde auf 1 bzw. 2 gesetzt.

Als Ergebnis ist festzuhalten, daß die Anzahl der Verkleinerungen nur einen sekundären Einfluß auf den Responsewert hat, da alle höheren Faktoren nur unnötige Auswertungen liefern. Insgesamt ist der Verkleinerungsfaktor von der Funktion abhängig, letztlich lieferten die Faktoren 2 und 5 aber die besten Responswerte. Bei Dimension 2 und 10 lieferte der Faktor 5 die besten Ergebnisse und bei den Dimensionen 3 bis 9 lieferte der Faktor 2 die besten Ergebnisse.

3.4.6. Untersuchung der Terminierungskriterien

Hier sollte untersucht werden, welchen Einfluss verschiedene Terminierungskriterien auf die Anzahl der Auswertungen und die Qualität des Ergebnisses haben. Konkret untersucht wurde die Anzahl der vorheriger Positionen, die mit der aktuellen verglichen werden um einen Stillstand festzustellen und die Anzahl der Bereichsverkleinerungen, die bei Stillstand erfolgen, bis der Algorithmus dann beendet wird. Getestet wurden auch hier verschiedene Funktionen mit unterschiedlicher Varianz der Rauschens. Eine Gemeinsamkeit viel sofort bei allen Simulationsläufen auf, nachdem ein Wert nahe des Optimums verhältnismäßig schnell gefunden wurde, gingen noch viele Auswertungen verloren bis der Algorithmus stoppte. Um dieses Verhalten zu minimieren empfahl es sich die Anzahl der Bereichsverkleinerungen auf 1 herunter zu setzen und auch die Zahl der verglichenen Vorgänger auf 1 zu setzen. Nur bei stark verrauschten Funktionen, lieferte eine höhere Anzahl bessere Ergebnisse, allerdings auf Kosten vieler Auswertungen. Bei nicht oder gering verrauschten Funktionen, erhielt man die besten Ergebnisse sogar,

wenn man nach einmaliger Verkleinerung des Suchbereichs und einem Folgedurchlauf den Algorithmus beendete. Fazit: Die Verschwendung von einigen Simulationsdurchläufen in der Größenordnung bis 80 läßt sich prinzipbedingt nicht vermeiden. Allein für einen Durchlauf bis es wieder zu einem Aufruf der Linearen Suche kommt, die die Position verändern könnte, werden beispielsweise bei dreidimensionalen Problemen ca. 20 Schritte gebraucht. (Je nach verwendetem Design etc.) Will man vermeiden die Optimierung zu früh abzubrechen, müssen einige dieser Schritte ausgeführt werden.

3.4.7. Stopkriterium der linearen Suche

Die Lineare Suche hat veränderliche Stopkriterien, die angeben, wie oft in eine Richtung, trotz verschlechternder Responsewerte gelaufen werden darf, bis der Algorithmus abbricht. Diesbezüglich wurde herausgefunden, daß Veränderungen der Stopkriterien beispielhaft an der Griewanks Funktion kaum Einfluß auf die Güte des Optimums haben, lediglich auf die Anzahl der Auswertungen, so daß sich die Fragestellung nach einem besten Wert für das Stopkriterium pauschal nicht beantworten läßt. Vorgeschlagen ist ein niedriger Wert oder gar ein Stopkriterium von 1, da das zu lösende Problem, aus lokalen Minima wieder herauszuspringen auch durch Anpassung anderer Klassen behoben werden kann.

3.4.8. Vergleich zwischen diskreten Faktoren und kontinuierlichen Faktoren

Da sich durch das Runden der Designpunkte die Qualität des Experimental Designs verschlechtert, sollte anhand von Benchmarkfunktionen untersucht werden, inwieweit sich die Ergebnisse von RSM bei Faktoren mit diskretem Wertebereich von den Resultaten von RSM mit kontinuierlichen Faktoren unterscheiden. Dafür wurden die Sphere, Step, Rastrigins und Ackleys Funktion jeweils für vier Faktoren in einem Suchbereich von -20 bis 20 und einer Ausdehnung des lokalen Suchbereichs von 5 untersucht. Bei der Sphere-Funktion waren die gefundenen Minima und die Anzahl der Auswertungen für RSM mit diskreten Faktoren fast genauso gut wie für das normale RSM. Bei schwierigeren Funktionen zeigte sich aber, daß RSM mit diskreten Faktoren bei etwa gleich guten Ergebnissen mehr Auswertungen brauchte, bzw. bei weniger Auswertungen etwas schlechtere Ergebnisse lieferte.

3.4.9. Ausdehnung des Suchbereichs bei diskreten Faktoren

Mit Hilfe von Benchmarkfunktionen sollte untersucht werden, bei welchen Größen des Suchbereichs RSM bei Faktoren mit diskretem Wertebereich noch sinnvolle Ergebnisse liefert. Dazu wurden mit der Sphere-, Step-, Rastrigins- und Ackleys-Funktion jeweils Suchbereichsgrößen von 0.5 bis 5 untersucht. Bei allen Funktionen lieferte RSM bei Ausdehnungen zwischen 5 und 3.5 noch gute Ergebnisse. Bei einer kleineren Ausdehnung war das gefundene Optimum sehr schlecht oder es wurde gar kein Fortschritt mehr erzielt. Abbildung 3.6 zeigt den Verlauf der Optimierung für die Ackleys-Funktion bei einer Ausdehnung des Suchbereichs von 3.5.

Die Ergebnisse dieser Untersuchung erscheinen plausibel, da sich bei einer Ausdehnung

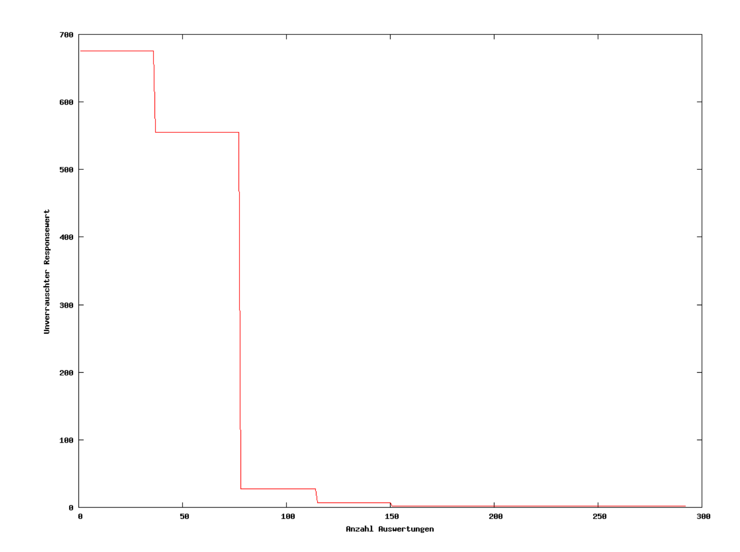


Abb. 3.5.: RSM-Leistungsstudien, Rastrigins-Funktion mit vier diskreten Faktoren

von 3.5 nach Runden insgesamt 5 unterschiedliche diskrete Werte im Suchbereich befinden, die Vorraussetzung für ein vernünftiges Experimental Design 2. Ordnung sind.

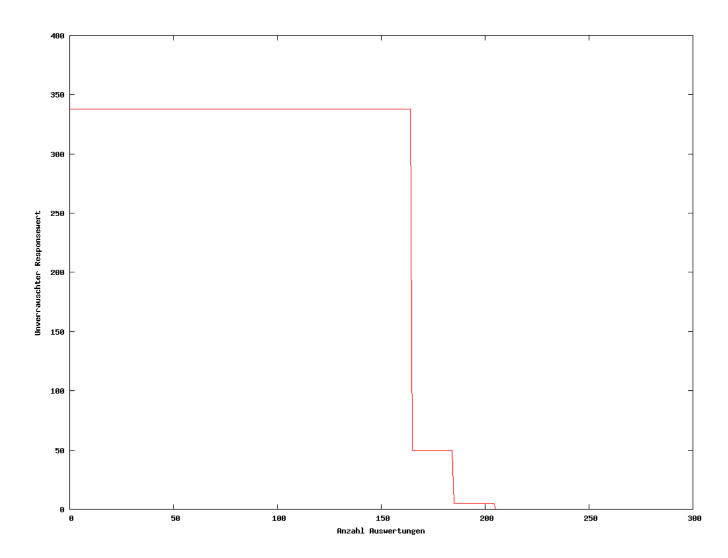


Abb. 3.6.: RSM-Leistungsstudien, Ackleys-Funktion mit Ausdehnung des Suchbereichs von 3.5

4. Evolutionäre Algorithmen

4.1. Theorie der Evolutionären Algorithmen

Evolutionäre Algorithmen dienen zur Lösung von Optimierungsproblemen wie zum Beispiel dem Traveling Salesman Problem. Es gibt nicht *den* evolutionären Algorithmus, sondern unterschiedliche Konzepte und Modelle definieren verschiedene Typen von evolutionären Algorithmen. Was aber alle gemeinsam haben ist, daß sie alle Konzepte und Vorgänge aus der natürlichen Evolution adaptieren. Ebenso wird auch die Terminologie aus der natürlichen Evolution übernommen, so daß man von "Populationen", "Individuen", "Selektion", "Mutation", "Rekombination" und "Fitneß" spricht. Die wesentlichen Faktoren, die zu einer Evolution führen, sind *Mutation*, *Selektion*, *Rekombination*. Bei der Selektion unterscheidet man zwischen Elternselektion und Umweltselektion. Dabei bezieht sich die Elternselektion auf die Auswahl der Paarungspartner und die Umweltselektion auf die Überlebenschancen ("Survival of the fittest").

Evolutionäre Algorithmen kombinieren den Computer als universelle Rechenmaschine mit dem Problemlösungspotential der natürlichen Evolution, indem im Computer ein Evolutionsprozeß künstlich nachgespielt wird, um ein Optimierungsproblem zu lösen. Dabei wird ein beliebiges abstraktes Objekt wie ein Organismus behandelt, indem es einen etwas vereinfachten evolutionären Zyklus (siehe Abbildung 4.1.1) durchläuft. Solch ein Objekt stellt meistens eine mögliche Lösung des betrachteten Problems dar. Im Zyklus wird dieses Objekt verändert, reproduziert und bewertet. Ein großes Problem stellt hierbei eine adäquate Modellierung dar, denn das Modell muß so beschaffen sein, daß man in möglichst kurzer Zeit einen möglichst guten Lösungskandidaten für das gegebene Problem berechnet. Besonders für Probleme, die in akzeptabler Zeit nicht exakt lösbar sind, stellen die evolutionären Algorithmen eine gute Lösung dar. Gegen problemspezifische effiziente Algorithmen haben sie natürlich keine Chance.

4.1.1. Der simulierte evolutionäre Zyklus

Katharina Balzer

Bei klassischen Optimierungsproblemen gibt es fast immer ein eindeutiges Bewertungsmaß für die Qualität eines Lösungskandidaten. Die Güte eines Lösungskandidaten kann einfach und eindeutig durch eine Bewertungsfunktion berechnet werden. Der Funktionswert dieser Bewertungsfunktion wird *Fitneß* genannt.

Betrachtet man das evolutionäre Wechselspiel zwischen Mutation/Rekombination auf der einen und Selektion als zielgerichtete Optimierung auf der anderen Seite, und überträgt dies auf Optimierungsprobleme, so kommt man schnell zum so genannten evolutionären Zyklus.

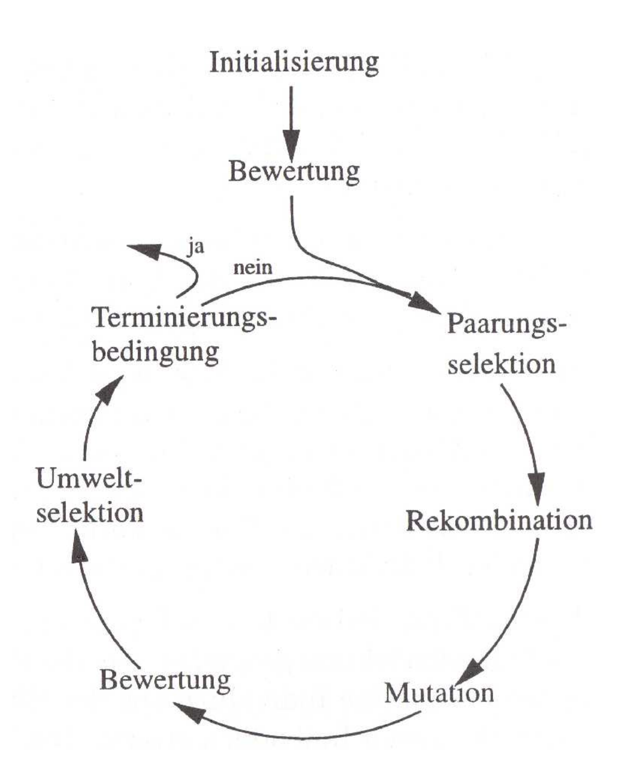


Abb. 4.1.: Der simulierte evolutionäre Zyklus

Die Grundidee dabei ist, daß als erstes eine Menge von Lösungskandidaten als Ausgangspunkt erzeugt wird, der anschließend eine simulierte Evolution durchläuft. D. h. die Lösungskandidaten pflanzen sich fort und unterliegen dabei einem gewissen Selektionsdruck. Wie schon zu Anfang erwähnt, spricht man hier in Analogie zur natürlichen Evolution von Individuen (Lösungskandidaten) und einer Population (Menge der Lösungskandidaten). Populationen sind im allgemeinen nicht sortiert.

Wenn man sich den evolutionären Zyklus anschaut, sieht man Mutation, Rekombination und Selektion als Komponenten, die aus der natürlichen Evolution übernommen wurden. Zusätzlich finden sich noch die Initialisierung, die Bewertung und ein Terminierungskriterium im Zyklus wieder. Im folgenden werden die einzelnen Schritte des evolutionären Zyklus näher erläutert.

Initialisierung Bei der Initialisierung wird eine erste Population von Lösungskandidaten erzeugt. Meistens geschieht dies, indem Lösungskandidaten zufällig ausgewählt werden. Manchmal ist aber auch durch das Optimierungsproblem eine initiale Population vorgegeben, oder Ergebnisse einer vorangegangenen Optimierung werden als Startwerte benutzt. Wenn für ein Problem keine bestimmte Struktur der Lösungskandidaten vorgegeben ist, werden sowohl die Struktur als auch die Werte zufällig gewählt.

Bewertung In der simulierten Evolution benötigt man eine Bewertungsfunktion, da die Individuen ja nicht in einer realen Umwelt leben, in der sie sich behaupten müssen. Sie dient als Vergleichskriterium zwischen den Indiviuen und führt so die Evolution zu besseren Lösungsräumen.

Selektion Um neue Individuen erzeugen zu können, müssen aus der bisherigen Population Eltern ausgewählt werden. Dies geschieht durch die so genannte Paarungsselektion, die eine Auswahl und Zuordnung der Anzahl von Nachkommen zu ihren Eltern darstellt. Die Paarungsselektion ist aber nur eine von zwei Selektionen, die im Laufe des evolutionären Zyklus stattfinden. Die zweite ist die Umweltselektion. Bei der Umweltselektion werden einzelne Individuen oder die gesamte Elternpopulation durch neue Individuen ersetzt. Dies muß geschehen, da die Populationsgröße meistens begrenzt ist. Die Unterscheidung der beiden Selektionstypen stammt aus der biologischen Evolution. Die meisten evolutionären Algorithmen nutzen jedoch nur eine von beiden Selektionen.

Rekombination Zwischen den Eltern, die bei der Paarungsselektion ausgewählt wurden, wird die Rekombination angewendet. Es entstehen ein oder mehrere neue Kindindividuen, indem das Genmaterial der Elternindividuen neu kombiniert wird. Hierbei findet man das Konzept der Vererbung aus der Biologie wieder.

Mutation Bei der Reproduktion des Genmaterials durch die Rekombination muß eine gewisse Variabilität in Form von Mutationsfehlern berücksichtigt werden. Dies geschieht durch die Anwendung von Mutationsoperatoren auf die Kindindividuen. Solch ein Mutationsoperator sollte im allgemeinen nur eine kleine Änderung am Genmaterial des Kindes vornehmen, da der Verlauf der Evolution wesentlich von der Vererbung des elterlichen Genmaterials abhängt. Außerdem würden große Veränderungen nur dazu führen, daß die so veränderten Individuen mit einer erhöhten Wahrscheinlichkeit bei der folgenden Selektion aussortiert würden. Ist die Mutation abgeschlossen, dann werden die Kindindividuen mit Hilfe der Bewertungsfunktion bewertet. Danach müssen die Kindindividuen in die bestehende Population intergriert werden. Dies geschieht dann mit der bereits erläuterten Umweltselektion.

Terminierungsbedingung Am Ende der simulierten Evolution wird, im Gegensatz zur natürlichen Evolution, überprüft, ob das gesteckte Ziel bereits erreicht wurde. Als Terminierungsbedingung kann zum Beispiel ein Schwellwert für die Fitneß des besten Individuums gewählt werden. Manchmal ist auch eine maximale Anzahl von Generationen vorgegeben, um den Berechnungsaufwand klein zu halten.

Um einen evolutionären Algorithmus der oben beschriebenen Form anwenden zu können, muß im Computer lediglich eine speicherbare Darstellung des Suchraums und eine Funktion zur Bewertung der Lösungskandidaten gefunden werden. Sonst gibt es keine weiteren Anforderungen an die Anwendbarkeit des Algorithmus. Dies ist eine der attraktivsten Eigenschaften von evolutionären Algorithmen.

4.1.2. Evolutionsstrategien und Genetische Algorithmen

Katharina Balzer, Julia Hielscher

Bei der Entwicklung der evolutionären Algorithmen haben sich drei Ausprägungen unabhängig voneinander entwickelt:

- Genetische Algorithmen
- Evolutionäres Programmieren und
- Evolutionsstrategien.

In diesem Abschnitt sollen nun zwei dieser Teilgebiete, die Evolutionsstrategien und die genetischen Algorithmen, näher betrachtet werden.

4.1.2.1. Evolutionsstrategien

Bei den Evolutionsstrategien werden Individuen mit reellwertigen Koordinaten benutzt. Primärer Operator ist der Mutationsoperator (siehe Abschnitt 4.1.3). Da die Rekombination (siehe Abschnitt 4.1.4) nur eine untergeordnete Rolle spielt, fällt somit der Mutation die Aufgabe der Erzeugung von Nachkommen zu. Selektion (siehe Abschnitt 4.1.5) tritt nur als Umweltselektion auf und nicht als Elternselektion. Die Eltern werden stattdessen zufällig gleichverteilt ausgewählt. Um den Optimierungserfolg zu verbessern, gibt es zudem sogenannte Anpassungsstrategien für die Schrittweite.

4.1.2.2. Genetische Algorithmen

Genetische Algorithmen sind neben den Evolutionsstrategien eine weitere weit verbreitete Form von evolutionären Algorithmen. Sie wurden in den USA von John Holland entwickelt. Üblicherweise arbeiten genetische Algorithmen auf Bitstrings, also im Suchraum $\mathbb{B}^n$. Der hauptsächliche Operator ist die Rekombination (siehe Abschnitt 4.1.4), während die Mutation (siehe Abschnitt 4.1.3) eine untergeordnete Rolle spielt. Meistens wird ein einfaches k-Punkt-Crossover verwendet mit $k \in \{1, 2\}$. Wenn Mutation verwendet wird, so geschieht dies oft in Form einer Standardbitmutation. Als Strategie zur Elternselektion kommt häufig die fitneßproportionale Selektion zum Einsatz, wohingegen die Umweltselektion implizit durch eine (μ, μ)-Strategie erfolgt. Die einzelnen Selektionsoperatoren sind in Abschnitt 4.1.5 näher erläutert.

4.1.3. Mutation

Peter Kissmann

In diesem Abschnitt werden zunächst einige verschiedene Mutationsoperatoren, wie sie beispielsweise auch in [22] beschrieben werden, vorgestellt, sowie für den letzten Operator noch einige Strategien, um die Parameter dieses Operators an den Suchraum und die Fitneßfunktion anzupassen.

4.1.3.1. k-Bit-Mutation

Die wohl einfachste Art bildet die k-Bit-Mutation. Mit einer gewissen Wahrscheinlichkeit werden in der Bitstring-Darstellung der Koordinaten jedes Individuums genau k Bits gekippt, also ein 1-Bit in ein 0-Bit umgewandelt und umgekehrt. Die grundlegendste Art bildet die 1-Bit-Mutation, bei der stets exakt ein Bit gekippt wird, aber es sind auch beliebige andere Anzahlen möglich.

4.1.3.2. Standardbit-Mutation

Ein weiterer Mutationsoperator, der ebenfalls auf der Bitstring-Darstellung der Koordinaten arbeitet, ist die Standardbit-Mutation. Hier wird nur eine Wahrscheinlichkeit angegeben, mit der jedes einzelne Bit gekippt wird. Als Standard wird hier der Wert $\frac{1}{n \cdot b}$ mit $n =$ Dimension und $b =$ Anzahl der Bits, die jede einzelne Koordinate kodieren. In diesem Fall wird im Erwartungswert wie bei der 1-Bit-Mutation ein Bit pro Individuum gekippt.

4.1.3.3. Reellwertige Mutation durch Addition eines Mutationsvektors

Neben diesen beiden Operatoren auf den Bitstrings gibt es auch noch einen, der auf den reellwertigen Koordinaten arbeitet: Die Mutation durch Addition eines Mutationsvektors. Hierbei wird ein zufälliger Vektor aus einer isotropen Normalverteilung gezogen und zu den Koordinaten der Individuen der Nachkommenpopulation hinzuaddiert. Die Koordinaten einer Population $Q = \{\mathbf{x}_1, \ldots, \mathbf{x}_\lambda\}$ werden also gemäß der Formel

$$\mathbf{x}'_i = \mathbf{x}_i + \sigma \mathbf{x}_i \tag{4.1}$$

mutiert. Die Mutationsvektoren $\mathbf{z}_i$ bestehen aus n (= Dimension) unabhängigen Komponenten, die aus einer Standardnormalverteilung gezogen werden. Das Skalar σ bestimmt die erwartete Entfernung der Nachkommen zum Mittelpunkt der Eltern; σ wird auch Mutationsstärke genannt.

Eine weitere Möglichkeit ist es, mit Winkeln zu arbeiten, um sich schneller dem Suchraum anzupassen. In diesem Fall spricht man auch von "korrelierter Mutation". Es werden insgesamt $n_\alpha = \frac{n \cdot (n-1)}{2}$ viele Rotationswinkel benötigt, um in Dimension n Drehungen auf allen Ebenen durchführen zu können. Diese Rotationswinkel werden sowohl bei der Rekombination, in der Regel analog zur Rekombination der Koordinaten, als auch bei der Mutation angepaßt. Bei der Mutation werden sie gemäß folgender Formel angepaßt:

$$\alpha'_m = \alpha_m + \beta \cdot N_m(0, 1) \tag{4.2}$$

Hierbei sind die α_m die einzelnen Rotationswinkel, β ein Faktor, der für die Anpassung gesetzt werden muß (ein empfohlener Wert für β ist 0,0873, was etwa 5° entspricht) und die $N_m(0, 1)$ Zahlen, die für jeden Winkel neu aus einer Standardnormalverteilung gezogen werden. Nun werden die Koordinaten der einzelnen Individuen gemäß

$$\mathbf{x}' = \mathbf{x} + \mathbf{\Delta}_{kor}$$

mutiert. Der korrelierte Zufallsvektor $\mathbf{\Delta}_{kor}$ ergibt sich als das Produkt von n_α Rotationsmatrizen $R_{p,q}(\alpha_m)$ und dem unkorrelierten Zufallsvektor $\mathbf{\Delta}_{unkor} = \sigma \cdot \mathbf{N}(0,1)$:

$$\mathbf{\Delta}_{kor} = \left(\prod_{p=1}^{n-1} \prod_{q=p+1}^{n} R_{p,q}(\alpha_m) \right) \cdot \mathbf{\Delta}_{unkor} \tag{4.3}$$

Die Rotationsmatrizen haben folgendes Aussehen:

$$R_{p,q}(\alpha_m) = \begin{pmatrix} 1 & 0 & \dots & \dots & \dots & \dots & 0 \\ 0 & \ddots & & & & & \vdots \\ \vdots & & \cos\alpha_m & \dots & -\sin\alpha_m & & \vdots \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ \vdots & & \sin\alpha_m & \dots & \cos\alpha_m & & \vdots \\ \vdots & & & & & \ddots & \vdots \\ 0 & \dots & \dots & \dots & \dots & \dots & 1 \end{pmatrix}$$

Die Matrizen entsprechen der n-dimensionalen Einheitsmatrix mit der Ausnahme, daß auf der Hauptdiagonalen an den Positionen (p,p) und (q,q) der Eintrag $\cos\alpha_m$, in Zeile p und Spalte q der Eintrag $-\sin\alpha_m$ und in Zeile q und Spalte p der Eintrag $\sin\alpha_m$ steht. Jede Multiplikation mit einer solchen Matrix entspricht einer Drehung in der p,q-Ebene um den Winkel α_m.

4.1.3.4. Strategien zur Adaption der Mutationsstärke

Falls die Mutationsstärke σ zu klein ist, verlangsamt sie den Fortschritt, ist sie hingegen zu groß, so führt sie zu Divergenz. Also wird ein dynamisches Schema benutzt, um die Evolutionsstrategie flexibel auf die lokalen Charakteristika der Zielfunktion anzupassen. Für Evolutionsstrategien mit mehreren Elternindividuen gibt es einige verschiedene Herangehensweisen, diese Anpassung durchzuführen.

4.1.3.4.1. Selbstadaption

Bei der Strategie der Selbstadaption der Mutationsstärken geht man davon aus, daß günstigere Mutationsstärken mit höherer Wahrscheinlichkeit erfolgreiche Nachkommen generieren als ungünstigere. In der Grundform werden die Mutationsstärken unverändert an die Nachkommen vererbt und die Wahl der günstigeren Mutationsstärken wird somit ein Nebenprodukt der Evolution.

Häufig werden die Mutationsstärken jedoch auch dem Rekombinationsprozeß mit unterzogen, in dem sie in der Regel genauso wie die Koordinaten aus unterschiedlichen Elternindividuen rekombiniert werden.

Auch in der Mutation werden oftmals die Mutationsstärken noch angepaßt: Es werden zwei Faktoren, τ_1 und τ_2 benötigt (Standardwerte für diese Faktoren sind $\tau_1 = \frac{1}{\sqrt{2 \cdot n}}$ und

$\tau_2 = \frac{1}{\sqrt{2 \cdot \sqrt{n}}}$). Dann arbeitet die Anpassung der Mutationsstärken folgendermaßen:

$$\sigma_i' = \sigma_i \cdot e^{\tau_1 \cdot N(0,1) + \tau_2 \cdot N_i(0,1)} \tag{4.4}$$

wobei $N(0,1)$ eine Zahl aus einer Standardnormalverteilung, die für alle Richtungen der Mutationsstärke dieselbe ist, sowie $N_i(0,1)$ eine Zahl aus einer Standardnormalverteilung, die für alle Richtungen der Mutationsstärke neu gezogen wird, darstellen.

4.1.3.4.2. Kumulative Adaption

Eine weitere Möglichkeit der Adaption der Mutationsstärken bildet die in [1] beschriebene kumulative Mutationsstärkenadaption: Es wird angenommen, daß, wenn die Mutationsstärke unter dem Optimum liegt, ausgewählte aufeinanderfolgende Schritte dazu tendieren, parallel zu sein, wenn die Mutationsstärke hingegen über dem Optimum liegt, diese Schritte zu Anti-Parallelität neigen. Intuitiv ist klar, daß es besser ist, einen größeren Schritt in eine Richtung zu machen, wenn mehrere kleinere Schritte in diese Richtung gehen, und daß die Schrittlänge zu groß ist, wenn sich aufeinanderfolgende Schritte gegenseitig auslöschen. Um parallele oder antiparallele Korrelationen der Fortschrittsvektoren sicher zu erkennen, müssen die Informationen einer gewissen Anzahl von Zeitschritten angesammelt werden.

Für die (μ, λ)-Evolutionsstrategie wird der akkumulierte Fortschrittsvektor $\mathbf{s}$ definiert durch $\mathbf{s}^{(0)} := \mathbf{0}$ und weiter

$$\mathbf{s}^{(t+1)} := (1-c)\,\mathbf{s}^{(t)} + \sqrt{\mu c\,(1-c)}\langle \mathbf{z} \rangle^{(t)}. \tag{4.5}$$

c ist hierbei eine Konstante, die angibt, wie weit die Erinnerung reicht, $\langle \mathbf{z} \rangle$ das arithmetische Mittel der Mutationsvektoren, die zu den für die nächste Elternpopulation selektierten Individuen korrespondieren; dieser Vektor wird Fortschrittsvektor genannt. $\sigma\langle \mathbf{z} \rangle$ verbindet die aufeinanderfolgenden Mittelpunkte der Populationen. Die Mutationsstärke wird gemäß

$$\sigma^{(t+1)} := \sigma^{(t)} \exp\left(\frac{\|\mathbf{s}^{(t+1)}\|^2 - n}{2Dn} \right) \tag{4.6}$$

aktualisiert. D ist hierbei ein Dämpfungsfaktor. In der Regel wird $c = \frac{1}{\sqrt{n}}$ und $D = \sqrt{n}$ gesetzt. Der Exponent in Gleichung 4.6 entspricht gerade der mittleren quadratischen Länge des akkumulierten Fortschrittsvektors, falls die aufeinanderfolgenden Fortschrittsvektoren voneinander unabhängig sind. Falls die quadratische Länge des akkumulierten Fortschrittsvektors kleiner als n ist, so wird die Mutationsstärke verringert, falls sie größer ist, wird sie erhöht.

4.1.4. Rekombination

Christian Horoba

4.1.4.1. Einleitung

In diesem Abschnitt werden Rekombinationsoperatoren allgemein erläutert. Als Grundlage wurde [2] verwendet.

Sei I die Menge aller Individuen. Ein Rekombinationsoperator ist eine probabilistische Funktion der Form:

$$r : I^m \to I,$$

wobei $m \geq 2$ gilt.

Man klassifiziert Rekombinationsoperatoren aufgrund der Struktur der betrachteten Individuenmenge I, auf der sie operieren.

Häufig bestehen die Individuen nur aus n-reellwertigen Komponenten, wobei n die Dimension des Suchraums bezeichnet. In diesem Fall gilt: $I = R^n$. Manchmal enthalten Individuen auch noch weitere Informationen, wie z. B. für jede Dimension separate Mutationsschrittweiten (Standardabweichungen) oder Rotationswinkel.

In dem Fall, daß die Individuen beide Informationen enthalten, gilt beispielsweise: $I = R^n \times R^n \times [-\pi, \pi)^{\frac{(n-1)\cdot n}{2}}$. Im Folgenden werden Rekombinationsoperatoren, die auf einer solchen Individuenmenge operieren, ES-Operatoren genannt.

Des Weiteren sind Rekombinationsoperatoren denkbar, die auf der Individuenmenge $I = \left(\{0,1\}^l\right)^n$ operieren, wobei l in diesem Fall die Anzahl der Bits, mit denen jede reelwertige Koordinate repräsentiert wird, bezeichnet. Diese Rekombinationsoperatoren werden im Folgenden GA-Operatoren genannt.

Die nächsten beiden Unterabschnitte gehen genauer auf verbreitete Instanzen der genannten Operatoren ein, wobei diese Auflistung natürlich nicht vollständig sein kann.

4.1.4.2. GA-Operatoren

Diese Operatoren operieren in der Regel auf zwei Individuen. Häufige Vertreter sind der k-Punkt-Crossover-Operator und der uniforme Crossover-Operator.

Der erstgenannte Operator wählt unabhängig uniform zufällig k Schnittstellen der Bitstrings und bildet die zu rekombinierenden Individuen auf das Individuum ab, das sich ergibt, indem man abwechselnd Stücke des ersten und des zweiten Individuums überträgt. Eine Eigenart dieses Operators stellt die Tatsache dar, daß die Wahrscheinlichkeit des Austauschs eines Bits von dessen Position abhängt. Um diese „Positionsabhängigkeit“ zu beseitigen, wurde vorgeschlagen, die Bitstrings beider Elterindividuen auf dieselbe Art und Weise zufällig zu permutieren, dann den k-Punkt-Crossover-Operator anzuwenden und die Bits des resultierenden Individuums mit der inversen Permutation wieder in die Ausgangsreihenfolge zu bringen. Des Weiteren können die Bitstrings an beliebigen Stellen durchtrennt werden. Die Einschränkung der möglichen Schnittstellen auf die zwischen den Bitrepräsentationen einzelner Koordinaten liegenden stellt eine weitere Variante dar.

Der zweite der hier vorgestellten Operatoren entscheidet bitweise unabhängig uniform zufällig, von welchem der beiden Elterindividuen das entsprechende Bit in den Bitstring des Nachkommens geschrieben werden soll.

Auf nahe liegende Weise sind Verallgemeinerungen auf mehr als zwei Elterindividuen denkbar.

4.1.4.3. ES-Operatoren

Die beiden Hauptvertreter dieser Operatoren werden diskrete und intermediäre Rekombinationsoperatoren genannt. Der Einfachheit halber werden diese beiden Varianten zuerst für den Fall, daß genau zwei Elterindividuen vorliegen, besprochen.

Bei der diskreten Rekombination wird für jede Komponente unabhängig uniform zufällig entschieden, von welchem der beiden Elterindividuen diese Information an den Nachkommen weitergegeben wird.

Bei der intermediären Rekombination werden die Komponenten beider Eltern jeweils gemittelt und dieser Wert anschließend an den Nachkommen weitergegeben.

Von beiden Operatoren existieren auch globale (panmiktische) Formen, die folgendermaßen arbeiten: Ein Individuum wird fest ausgewählt. Für jede Komponente dieses Individuums wird ein weiteres Individuum selektiert. Im Anschluß wird die betreffende Komponente des Nachkommens auf die oben beschriebene Art berechnet, wobei stets das fest gewählte Individuum und ein weiteres Individuum in die Berechnung eingehen.

Weitere Varianten des intermediären Rekombinationsoperators wurden vorgestellt. So ist es z. B. denkbar, die Komponenten der Elterindividuen zu gewichten und nicht einfach deren Mittelwert zu verwenden. Desweiteren sind Mischformen der vorgestellten Konzepte denkbar. Eine Variante sieht vor, die Objektvariablen paarweise diskret und die Strategieparameter (Mutationsschrittweiten und Rotationswinkel) global intermediär zu rekombinieren.

4.1.5. Selektion

Katharina Balzer

Selektion nennt man den Prozeß des Auswählens bestimmter Individuen aus einer oder mehreren Populationen. Im allgemeinen unterscheidet man zwei Fälle: die Elternselektion und die Umweltselektion. Bei letzterer unterscheidet man weiter in Komma- und Plusselektion. Wählt man die Individuen nur aus der Population der Nachkommen, so spricht man von einer *Komma-Selektion*, wählt man aber sowohl aus der Population der Eltern als auch aus der der Nachkommen, so spricht man von einer *Plus-Selektion*. Alle im folgenden vorgestellten Operatoren können sowohl für die Eltern- als auch für die Umweltselektion verwendet werden. Ein weiterer Parameter bei der Selektion ist, ob man mit oder ohne Zurücklegen der Individuen selektieren möchte. Dabei bedeutet „mit Zurücklegen", daß jedes Individuum mehrfach in die neue Population aufgenommen werden kann. Zieht man „ohne" Zurücklegen, so wird jedes Individuum maximal einmal in die neue Population übernommen. Dies bedeutet, daß in diesem Fall nicht

mehr Individuen in die neue Population übernommen werden können, als in den alten Populationen vorhanden sind.

4.1.5.1. Schnitt-Selektion

Bei dieser sehr einfachen Selektionsform werden immer deterministisch die k Individuen mit der besten Fitneß aus der Population gewählt.

4.1.5.2. uniforme Selektion

Bei der uniformen Selektion werden die Individuen uniform zufällig ausgewählt. Die Fitneß der Individuen spielt dabei keine Rolle.

4.1.5.3. Turnierselektion

Dieser Selektionsoperator benötigt einen zusätzlichen Parameter, die Turniergröße. Der Parameter gibt an, wieviele Individuen jeweils an einem Turnier beteiligt sind. Für die endgültige Auswahl eines Individuums findet ein Turnier statt, für das so viele Individuen uniform zufällig gezogen werden, wie die Turniergröße vorgibt. Das Individuum mit der besten Fitneß gewinnt jeweils das Turnier und wird endgültig in die neue Population übernommen. Insgesamt werden soviele Turniere veranstaltet, bis die neue Population vollständig gefüllt ist, oder die vorgegebene Anzahl Eltern für die Rekombination gezogen sind. Zieht man bei der Umwelt-Komma-Selektion ohne Zurücklegen, so sollte die Turniergröße nicht größer als die Kinderpopulation sein. Dies gilt analog für die Umwelt-Plus-Selektion und die Kinder- und Elternpopulation, sowie für die Elternselektion und die Elternpopulation.

4.1.5.4. fitneßproportionale Selektion

Bezeichne $f(a)$ die Fitneß des Indiviuums a. Dann wird bei der fitneßproportionalen Selektion das Individuum a mit der Wahrscheinlichkeit $f(a)/\sum_{x\in P} f(x)$ gewählt, wobei P die Population ist, aus der gezogen wird. Nachteil dieses Verfahrens ist, daß alle Fitneßwerte immer positiv sein müssen, da sonst keine Verteilung definiert wird, gemäß der gezogen werden kann.
Eine Variante der fitneßproportionalen Selektion ist die *Rangselektion*. Hierbei werden alle Individuen anhand ihrer Fitneß geordnet und erhalten somit eine eindeutige Position in der Rangfolge aller Individuen. Sei 1 das schlechteste und n das beste Individuum in der Rangfolge und sei $pressure \in [1,2]$ der Selektionsdruck. Dann berechnet sich die neue Fitneß eines jeden Indivduums wie folgt:

$$f(x) = 2 - pressure + 2 \cdot (pressure - 1) \cdot \frac{x-1}{n-1}$$

Anhand der neuen Fitneßwerte wird nun eine Verteilung auf den Individuen berechnet, gemäß der die Individuen dann gezogen werden. Der Vorteil dieser Variante ist, daß auch negative Fitneßwerte vorkommen dürfen.

4.1.6. Einbindung statistischer Ranking & Selection-Verfahren

Peter Kissmann

Bei Evolutionären Algorithmen gibt es insgesamt vier Stellen, an denen sich Ranking & Selection-Verfahren, wie sie in Kapitel 6 beschrieben sind, integrieren lassen:

1. **Elternselektion:** Hier müssen unter Umständen (außer bei der uniformen Selektion) verschiedene Individuen miteinander verglichen werden und entschieden werden, welche die besten sind. Daher ist hier eine Anwendung der R&S-Verfahren sinnvoll.

2. **Umweltselektion:** Wie bei der Elternselektion, werden auch innerhalb der Umweltselektion verschiedene Individuen miteinander verglichen und die besten sollen für die neue Elternpopulation ausgewählt werden. Somit bietet sich auch hier ein R&S-Verfahren an.

3. **Verwaltung der Elite-Population:** Da der Algorithmus auch eine Population der Individuen verwaltet, die als die zum entsprechenden Zeitpunkt besten gelten (die Elite-Population), kann auch hier problemlos ein R&S-Verfahren eingebunden werden, um die einzelnen Individuen der drei Populationen miteinander zu vergleichen.

4. **Finalselektion:** Am Ende des Algorithmus muß schließlich ein bestes Individuum aus der Elite-Population ausgewählt werden. Um dieses bestimmen zu können, kann ebenfalls R&S verwendet werden. Somit ist es dann durchaus möglich, die Individuen der Elite-Population weiteren Auswertungen zu unterziehen und möglicherweise sogar ein anderes Individuum als bestes zu identifizieren, da durch die zusätzlichen Auswertungen das Rauschen weiter abgeschwächt wird.

4.1.7. Problemspezifische Verfahren

Julia Hielscher

4.1.7.1. Optimieren mit Randbedingungen

Randbedingungen schränken den durch die Zielfunktion bestimmten Lösungsraum ein. Bei der Optimierung sollte also nicht nur die optimale Lösung bezüglich der Bewertungsfunktion gefunden werden, sondern zusätzlich sollte auch auf die Einhaltung eventueller Randbedingungen geachtet werden.
Es gibt verschiedene Verfahren um Randbedingungen zu berücksichtigen.

1. Es wird auf dem unbeschränkten Suchraum optimiert und durch zusätzliche Maßnahmen das Vorkommen von ungültigen Individuen vermieden.
   Eine Technik um die ungültigen Individuen zu vermeiden ist die Methode Kindstod. Hierbei wird jedes neu erzeugte Individuum, das eine Randbedingung nicht erfüllt, sofort gelöscht. Problematisch kann es werden wenn viele gelöscht werden müssen, da es dann lange dauert bis genügend gültige Individuen erzeugt wurden.

Zudem ist es manchmal sinnvoll ungültige Individuen als Eltern zu zulassen, da sonst bestimmte gültige Kandidaten nicht gefunden werden können.

2. Ungültige Individuen sind zugelassen, werden aber in der Evolutionssimulation benachteiligt.
   Eine Möglichkeit besteht darin, daß die ungültigen Individuen schlechter bewertet werden als die gültigen. Dies kann durch Straffunktionen geschehen.

4.1.7.2. Verrauschte Zielfunktionen

Bewertungen können nicht immer exakt vorgenommen werden, sondern sind manchmal fehlerbehaftet. In diesen Fällen spricht man von verrauschten Zielfunktionen. Im folgenden Abschnitt wollen wir unter Rauschen eine relativ kleine Abweichung von einem tatsächlichen Gütewert verstehen.
Verrauschte Zielfunktionen haben meistens negative Auswirkungen, da die Zuverlässigkeit und Objektivität der Bewertungsfunktion nicht mehr gegeben ist. So kann es sein, daß gute Individuen als schlecht identifiziert werden oder auch umgekehrt. Es ist also bei verrauschten Zielfunktionen schwieriger die guten Individuen zu finden.
Somit stellt sich die Frage, wie die negativen Auswirkungen des Rauschens minimiert werden können. Eine Möglichkeit besteht darin, jedes Individuum mehrfach zu bewerten und den Mittelwert zu berechnen.

4.2. Programmbeschreibung

4.2.1. Anforderungsbeschreibung

Julia Hielscher

Im folgenden Abschnitt sollen nun die Anforderungen der EA-Gruppe behandelt werden. Implementiert werden sollte ein evolutionärer Algorithmus, der über verschiedene Möglichkeiten zur Initialisierung, Selektion, Rekombination, Mutation, Auswertung und Terminierung verfügt. Das Programm sollte sowohl unter Linux / Unix als auch unter Windows lauffähig sein. Zur Auswertung der Individuen und Berechnung der Fitneßwerte sollte eine Anbindung an verschiedene Benchmarkfunktionen, die in Abschnitt 2.4 beschrieben werden, sowie an die Simulationswerkzeuge APNN (siehe Abschnitt 8.2) und ProC/B (siehe Abschnitt 8.1) erfolgen.
Besonderen Wert wurde auf die Modularität des entstehenden Programmes gelegt. Einzelne Ausprägungen der verschiedenen Operatoren sollten beliebig miteinander kombinierbar sein. Zudem sollte auch problemlos das Einfügen neuer Operatorausprägungen möglich sein.
Ein weiterer Punkt waren die beiden Ausprägungen der evolutionären Algorithmen, Evolutionsstrategien und genetische Algorithmen, die in Abschnitt 4.1.2 vorgestellt wurden, und beide in der Implementierung enthalten sein sollten. Hierbei wurde entschieden, die Benutzer nicht in der Wahl der Operatoren einzuschränken. Es sollten keine festen Vorgaben gemacht werden, die bei Wahl einer der beiden Algorithmentypen nur solche

Operatorarten zulassen, die für diesen Algorithmentyp typisch sind. Stattdessen sollte auch hier eine freie Kombinierbarkeit gegeben sein.
Die freie Kombinierbarkeit der Algorithmentypen warf das Problem auf, daß Evolutionsstrategien vorwiegend auf reellen Zahlen arbeiten, während genetische Algorithmen meistens eine binäre Darstellung erfordern. Somit mußte zu jedem Zeitpunkt des Programmablaufes die Möglichkeit bestehen, die Individuenkoordinaten als reelle und als binäre Zahlen zu erhalten.
Die Einbindung der Simulationswerkzeuge brachte das Problem mit sich, daß mit stochastischen Schwankungen der Simulationsausgaben zu rechnen ist. Um dem entgegen zu treten, sollten mehrfache Auswertungen eines Individuums möglich sein. Zudem sollte eine Möglichkeit gefunden werden, bei verrauschten Ausgaben potentiell gute Individuen zu finden. Zur Unterstützung dieses Suchprozesses bot es sich an, während des Algorithmusdurchlaufes nicht nur ein bestes Individuum zu speichern, sondern sich eine Menge von Individuen mit guten Fitneßwerten zu merken und diese potentiell guten Lösungskandidaten eventuell weiteren Auswertungen zu unterziehen.
Im Anschluß an den Algorithmusdurchlauf sollte der Benutzer zudem die Möglichkeit haben, die Ergebnisse in einer separaten Datei einzusehen. Hierbei wurde nicht nur das am Ende als bester Lösungskandidat ausgewählte Individuum als relevant angesehen. Die gesamte Menge der besten Individuen, die während des Durchlaufes entsteht und variiert, sollte festgehalten werden.
Wie alle diese Anforderungen umgesetzt wurden, ist in Abschnitt 4.2.2 zu lesen.

4.2.2. Klassendiagramm und Beschreibung des Aufbaus

Igor Gudovsikov, Peter Kissmann

In Abbildung 4.2 ist das Klassendiagramm der evolutionären Algorithmen dargestellt. Bei der Entwicklung der Klassen wurde darauf geachtet, daß sie möglichst erweiterbar sind. Daher wurde oftmals Vererbung eingesetzt.

Der Ablauf des Programms beginnt, indem in der *Controller*-Klasse alle Instanzen erzeugt werden. Die Informationen, welche Klassen mit welchen Einstellungen erstellt werden sollen, werden aus der *BlackBox*, die den Aufruf wiederum an die *Whitebox* weiterleitet, ausgelesen. In der Mitte der Abbildung erkennt man die zentrale Klasse *EvolutionaryAlgorithm*, in der sich der Hauptzyklus des Programms befindet. In diesem Zyklus werden alle standardmäßigen EA-Schritte, also Initialisierung, Eltern-Selektion, Rekombination, Mutation, Umwelt-Selektion und Terminierung durchgeführt. Um es zu ermöglichen, diese Klassen zu erweitern, wurden sie zuerst als Interfaces entwickelt. Im Weiteren wurden von jedem dieser Interfaces die Klassen mit implementierten Methoden abgeleitet (in dem Diagramm aus Übersichtsgründen nicht zu sehen). Das macht das Programm flexibel und ermöglicht es dem Benutzer, verschiedene Konfigurationen von EA-Schritten auszuprobieren und dabei für bestimmte Problemstellungen die besten zu finden.

Neben diesen Operatoren wurden noch insgesamt drei Populationen in den *EvolutionaryAlgorithm* eingebunden. Diese stellen die Eltern-, Nachkommen sowie Elite-Population dar und verwalten eine Anzahl von Individuen. Diese wiederum verfügen neben den Ko-

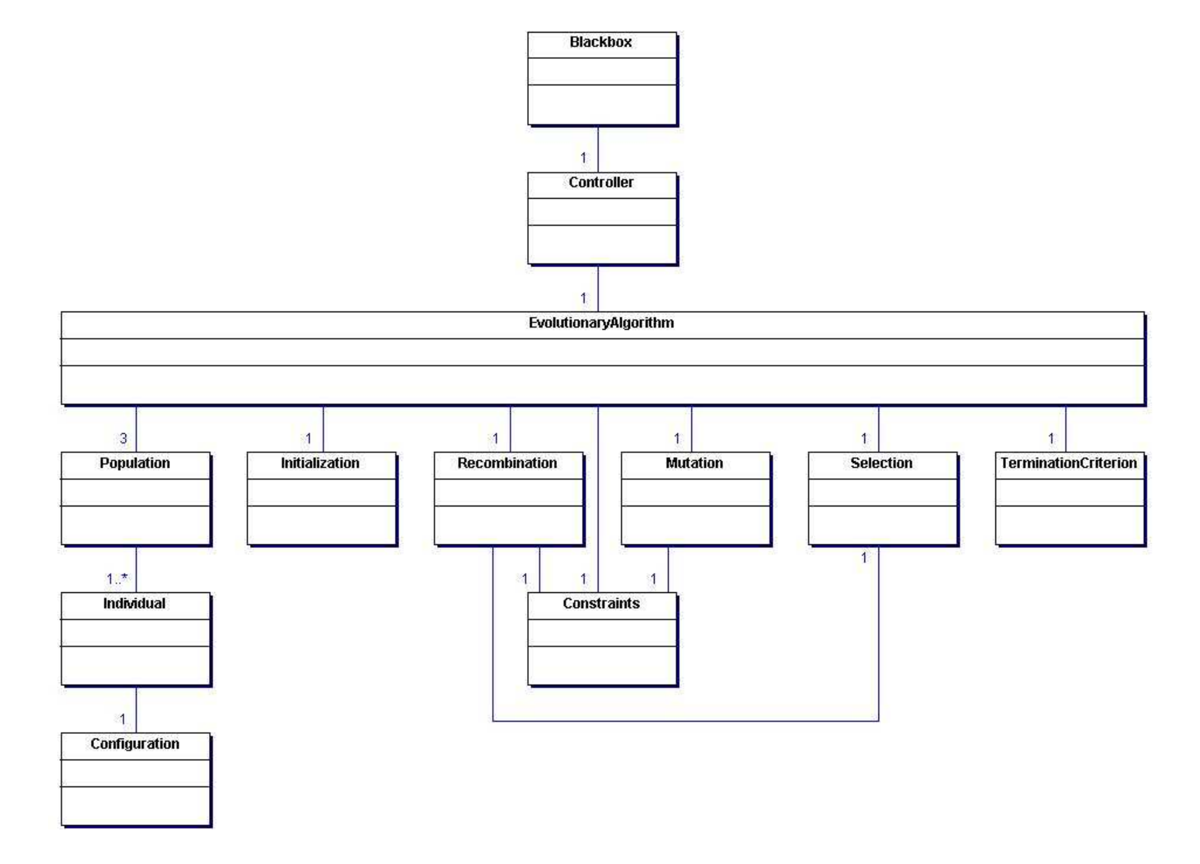


Abb. 4.2.: EA Klassendiagramm

ordinaten (in binärer und reellwertiger Darstellung) sowie einigen weiteren Attributen auch jeweils über eine *Configuration*, die sämtliche Auswertungen, die für ein Individuum durchgeführt wurden, speichert.

Desweiteren gibt es noch ein *Constraints*-Objekt. Hier werden die Nebenbedingungen verwaltet und die *Recombination* und *Mutation* können jederzeit, nachdem sie neue Individuen erzeugt haben, überprüfen, ob diese im gültigen Bereich liegen und entsprechend handeln.

Eine ausführliche Beschreibung der Klassen wird in Abschnitt 4.3 gegeben.

4.3. Klassenbeschreibung

4.3.1. Constraints

Katharina Balzer

Die Klasse *Constraints* realisiert die Nebenbedingungen, die an ein Individuum gestellt werden. Es können nur linke und rechte Grenzen für die Koordinaten eines Individuums angegeben werden. Die Nebenbedingungen werden in zwei Arrays *upper* und

lower gespeichert. Die Indexposition der Arrays korrespondiert zu der Indexposition der entsprechenden Koordinate im Koordinatenarray. Desweiteren wird in einem zusätzlichen Array gespeichert, ob jede Koordinate diskret oder kontinuierlich ist.
In der Klasse *Constraints* ist eine Methode *check* realisiert, die überprüft, ob ein Individuum seine Nebenbedingungen einhält oder nicht. Ist dies nicht der Fall, so wird im Individuum ein boolesches Attribut *illegal* auf *true* gesetzt.

4.3.2. Controller

Igor Gudovsikov, Christian Horoba

Die *Controller*-Klasse ist die Schnittstelle zwischen den Klassen, die die evolutionären Algorithmen realisieren, und der *BlackBox*.

Die zentrale Aufgabe der *Controller*-Klasse ist die Erzeugung aller für einen konkreten evolutionären Algorithmus benötigten Objekte. Im Allgemeinen funktioniert das folgendermaßen: Zuerst werden in der *run*-Methode der Klasse *Controller* die Parameter aus der *BlackBox* ausgelesen. Im Anschluß werden das Vorhandensein und die Korrektheit aller notwendigen Parameter geprüft. Waren diese Tests erfolgreich, werden die Objekte konstruiert, danach die *startCycle*-Methode des *EvolutionaryAlgorithm* aufgerufen und der Kontrollfluß abgegeben. Sobald alle Berechnungen abgeschlossen sind, werden der Ablauf an die *Controller*-Klasse zurückgegeben und die Objekte anschließend gelöscht, um den Speicher freizugeben.

Die relevanteste Methode der *Controller*-Klasse ist die *run*-Methode. Ein Aufruf startet den Hauptzyklus des durch die in der *BlackBox* hinterlegten Parameter spezifizierten evolutionären Algorithmus. Die *run*-Methode erwartet einen Zeiger auf die *BlackBox* und eine Referenz auf einen Vektor, der die berechnete Faktorenbelegung enthalten wird. Diese Methode ist aus mehreren – im Prinzip ähnlich aufgebauten – Teilen zusammengesetzt. Jeder Teil ist für die Erzeugung von Objekten der folgenden Klassen

- *Constraints*
- *EvolutionaryAlgorithm*
- *Individual*
- *Initialization*
- *Mutation*
- *Population*
- *RankingAndSelection*
- *Recombination*
- *Selection*
- *TerminationCriterion*

verantwortlich. Von den meisten Klassen wird jeweils eine Instanz konstruiert, wobei folgende Ausnahmen gelten: Von der Klasse *Individual* wird die für die Populationen benötigte Anzahl Instanzen erzeugt, von der Klasse *Population* werden drei Instanzen erzeugt (Populationen von Elternindividuen, Nachkommenindividuen und Eliteindividuen), von der Klasse *RankingAndSelection* werden zwei bis vier Instanzen erzeugt (ggf. Verfahren für die Elternselektion und die Umweltselektion und Verfahren für die Verwaltung der Elitepopulation und die Endauswahl) und von der Klasse *Selection* werden zwei Instanzen erzeugt (Elternselektion und Umweltselektion).

Um den Ablauf in jedem Teil genauer zu beschreiben, betrachten wir exemplarisch die Erzeugung einer Instanz der Klasse *Mutation*. Zuerst werden mithilfe der *GetParameter*-Methoden die benötigten Parameter aus der *BlackBox* ausgelesen und die Werte in lokalen Variablen gespeichert (falls einer der Parameter nicht vorhanden oder fehlerhaft ist, wird eine Fehlermeldung ausgegeben, die bisher konstruierten Objekte gelöscht und die Methode beendet). Danach wird anhand des Werts des MUTATION-Parameters entschieden, welche Art der Mutation spezifiziert wurde oder genauer gesagt, von welcher Unterklasse der *Mutation*-Klasse eine Instanz benötigt wird. Für die Erzeugung einer Instanz einer Unterklasse werden einige spezifische Parameter benötigt. Diese werden ebenfalls auf Vollständigkeit überprüft. Jede Unterklasse der *Mutation*-Klasse verfügt über eine statische *check*-Methode, die überprüft, ob alle Parameter im zulässigen Wertebereich liegen (falls das nicht der Fall ist, wird ebenfalls eine Fehlermeldung ausgegeben und die *run*-Methode nach Durchführung der oben geschilderten Aufräumarbeiten verlassen). Anschließend werden schließlich alle Parameter an den Konstruktor übergeben und eine Instanz der Klasse erzeugt. Am Ende jedes Teils wird die *insert*-Methode des *EvolutionaryAlgorithm* aufgerufen, um in den entsprechenden Attributen die Adressen der konstruierten Objekte zu hinterlegen.

Die Konstruktion von Instanzen funktioniert für alle oben aufgelisteten Klassen auf ähnliche Weise.

4.3.3. EvolutionaryAlgorithm

Julia Hielscher, Peter Kissmann

Die Klasse *EvolutionaryAlgorithm* realisiert den evolutionären Zyklus. In der Methode *startCycle* wird zunächst die gewünschte Initialisierungsmethode aufgerufen. Anschließend beginnt eine Schleife, die so lange wiederholt wird, bis die Terminierungsbedingung erfüllt ist. Innerhalb der Schleife werden nacheinander Rekombination, Mutation, Umweltselektion und gegebenenfalls Mutationsstärkenadaption aufgerufen. Um die verschiedenen Operatoren benutzen zu können, müssen sie der Klasse bekannt sein. Deshalb werden sie nach der Erzeugung durch insert-Methoden hinzugefügt und können durch get-Methoden auch von anderen Klassen abgerufen werden. Die Klasse *EvolutionaryAlgorithm* fungiert somit auch als Schnittstelle zwischen den verschiedenen evolutionären Operationen.
Auch für die Verwaltung der Elite-Population ist die Klasse *EvolutionaryAlgorithm* verantwortlich. Die Methode *updateBestIndividuals* fügt neue Individuen in die Population ein, solange diese noch nicht voll ist. Wenn die Population gefüllt ist, können enthaltene

Individuen durch neue bessere Individuen ersetzt werden. Hierbei wird dann ein Ranking & Selection-Verfahren angewendet, um zu bestimmen, welche Individuen tatsächlich die bisher besten sind und somit in der Elite-Population bleiben sollten. Hierdurch können natürlich zusätzliche Auswertungen angestoßen werden.
Zudem ist die Klasse *EvolutionaryAlgorithm* für die Erzeugung einer Ergebnisdatei zuständig. In dieser werden während des Algorithmusdurchlaufes nach jeder Iteration die Individuen der Elite-Population eingetragen. Zu den festgehaltenen Werten gehören die Koordinaten als reellwertige Zahlen und in Binärdarstellung, die Fitneß und das Konfidenzintervall des Individuums. Hat der Algorithmus terminiert, wird außerdem das beste Individuum aus der Elite-Population bestimmt. Auch die Bestimmung des besten Individuum funktioniert wieder über ein Ranking & Selection-Verfahren. Dazu werden alle Individuen der Elite-Population miteinander verglichen, was - je nach Verfahren - auch weitere Auswertungen nach sich ziehen kann, das beste von diesen dann zurückgeliefert und seine Werte ebenfalls in die Ergebnisdatei eingetragen.
Desweiteren ist die Klasse *EvolutionaryAlgorithm* dafür zuständig, ein Array aller Individuen zu speichern, in dem diese geordnet enthalten sind. Nach Erzeugung aller Populationen wird die Methode *updateIndsOrdered* aufgerufen, mit der die Individuen geordnet in das Array eingefügt werden. Die Arrayposition entspricht der eindeutigen Nummer, die jedes Individuum erhalten soll. Dieses Verfahren ist notwendig, um feststellen zu können, welche Individuen zueinander korrespondieren.

4.3.4. Individual

Julia Hielscher

Die Klasse *Individual* repräsentiert die Lösungskandidaten für ein Optimierungsproblem. Um die Individuen selektieren, rekombinieren, mutieren und bewerten zu können, verfügt die Klasse *Individual* über zahlreiche Attribute. So werden vor allem die Koordinaten als Array gespeichert. Dies geschieht in Form von reellen Zahlen und als Binärdarstellung. Diese doppelte Speicherung dient dazu, sowohl Evolutionsstrategien, die für die reellwertige Darstellung bekannt sind, als auch Genetische Algorithmen, die vorwiegend auf der Binärdarstellung arbeiten, benutzen und eventuell sogar mischen zu können. Für die Umrechnung speichert die Klasse *Individual* noch die Anzahl Bits, die pro reellwertiger Zahl für die binäre Darstellung verwendet werden sollen. Auch die Mutationsstärken und Rotationswinkel, die zu jedem Individuum gehören, werden in beiden Formen gespeichert. Wird ein Wert in einer der beiden Darstellungen geändert, so erfolgt sofort eine Anpassung der jeweils anderen Form. Die Binärdarstellung kann zusätzlich noch graykodiert werden, was sich als effizientere Darstellung für den Suchprozeß herausgestellt hat.
Zusätzlich speichert ein Individuum noch weitere Werte, die den anderen Klassen helfen, ihre Aufgaben zu erfüllen. Zum Beispiel:

- die Anzahl Dimensionen des Suchraumes,
- die Angabe, ob das Individuum legal ist, also alle Koordinaten innerhalb des zulässigen Bereiches liegen und

- ein Configuration-Objekt, welches die Faktoren und Ergebnisse der bis zum aktuellen Zeitpunkt durchgeführten Auswertungen speichert.

Die Daten werden vorwiegend von anderen Klassen benutzt und können somit über get- und set-Methoden abgerufen und neu gesetzt werden.
Desweiteren verfügt jedes Individuum aus den verschiedenen Populationen über eine Nummer. Diese dient dazu, das Individuum eindeutig zu indentifizieren. Somit ist es möglich, für jedes Individuum ein Array zu speichern, in dem die Individuen eingetragen sind, die sich von dem aktuellen Individuum nicht unterscheiden. Vor allem wenn ein Individuum in die Population der besten Individuen aufgenommen wird, ist dies wichtig, da somit verhindert werden kann, daß diese Population der Besten nur aus Kopien eines Individuums besteht. Indem überprüft werden kann, ob zwei Individuen zueinander korrespondieren, kann sichergestellt werden, daß die Population der besten Individuen vielfältig ist.

4.3.5. Initialization

Julia Hielscher

Die Klasse *Initialization* dient als Oberklasse für die verschiedenen Initialisierungsmethoden. Bei Letzteren handelt es sich um *InitUniform*, *InitBounds*, *InitDirect* und *InitRSM*. Die Initialisierung sorgt dafür, daß die Individuen der ersten Elternpopulation erzeugt werden. Initialisert werden die Koordinaten, die Mutationsstärken und die Rotationswinkel der Individuen. Es muß stets beachtet werden, daß die erzeugten Individuen legal sind, also nicht außerhalb des zulässigen Bereiches liegen.

4.3.5.1. InitBounds

Die Klasse *InitBounds* kann benutzt werden, wenn die Anfangsindividuen nicht über den gesamten Suchraum verstreut sein sollen. Mit *InitBounds* können zusätzlich zu den üblichen Nebenbedingungen für jede Dimension weitere Grenzen eingegeben werden, innerhalb derer sich die Koordinaten der Individuen der ersten Elternpopulation befinden sollen. Diese zusätzlichen Grenzen müssen natürlich zwischen den Nebenbedingungsgrenzen liegen und somit den Initialisierungsbereich weiter eingrenzen, da ansonsten ungültige Individuen erzeugt werden könnten. Nützlich ist diese Initialisierungsmethode vor allem, wenn bestimmte Effekte beobachtet werden sollen oder wenn bereits der Bereich für das Optimum vor dem Start des evolutionären Algorithmus eingegrenzt werden konnte.
Um nun also die Koordinaten der Individuen zu initialisieren, werden uniforme Zufallszahlen aus dem Bereich gezogen, der von den zusätzlichen Grenzen vorgegeben ist. Für die Erzeugung der Rotationswinkel muß beachtet werden, daß sie nur aus dem Bereich von $-\pi$ bis π gewählt werden dürfen. Zusätzlich wird der Bereich für die Mutationsstärken eingegrenzt, so daß diese nicht zu groß werden. Sehr große Mutationsstärken würden dafür sorgen, daß im weiteren Verlauf häufig Individuen entstehen, deren Koordinaten nicht mehr im zulässigen Bereich sind. Um dies zu vermeiden werden die Werte für die Mutationsstärken zufällig aus dem Bereich 0 bis $(obereGrenze - untereGrenze)/4$

gewählt, wobei die Grenzen den vom Benutzer vorgegebenen „Bounds“ für jede Dimension entsprechen.

4.3.5.2. InitDirect

Die Klasse *InitDirect* gibt dem Benutzer die Möglichkeit die Koordinaten, Mutationsstärken und Rotationswinkel der Anfangsindividuen selber vorzugeben. Dabei sind drei verschiedene Arten möglich:

1. Koordinaten, Mutationsstärken und Rotationswinkel werden vom Benutzer vorgegeben.
2. Nur die Koordinaten sind vom Benutzer vorgegeben. Die Rotationswinkel und Mutationsstärken werden zufällig erzeugt. Beachtet werden muß wieder wie bei *InitBounds*, daß die Rotationswinkel nur aus dem Bereich von $-\pi$ bis π gewählt werden dürfen und daß die Mutationsstärken nicht zu groß sein sollen. Deshalb werden die Mutationsstärken zufällig aus dem Bereich 0 bis $(obereGrenze - untereGrenze)/4$ gewählt, wobei die Grenzen den vom Benutzer vorgegebenen Nebenbedingungsgrenzen für jede Dimension entsprechen.
3. Nur die Koordinaten sind vom Benutzer vorgegeben. Die Rotationswinkel werden zufällig aus dem Bereich $-\pi$ bis π erzeugt. Es wurde aber eine kumulative Mutationsstärkenadaption gewählt. Deshalb werden die Mutationsstärken auf 0 gesetzt.

4.3.5.3. InitUniform

Die Klasse *InitUniform* erzeugt die Anfangspopulation uniform zufällig. Beachtet werden muß nur, daß die Koordinaten der Individuen zwischen den vom Benutzer vorgegebenen Grenzen für eine Dimension liegen und daß die Rotationswinkel nur aus dem Bereich von $-\pi$ bis π gewählt werden dürfen. Für die Mutationsstärken gelten die bei *InitBounds* beschriebenen Bedingungen, wobei die Grenzen jedoch in diesem Fall nicht die „Bounds“, sondern die vorgegebenen Bereichsgrenzen für jede Dimension sind.

4.3.5.4. InitRSM

Die Klasse *InitRSM* erzeugt die Anfangspopulation zunächst uniform zufällig. Jedes Individuum wird aber zusätzlich, durch einen Aufruf der Response Surface Methode, in ein lokales Optimum verwandelt. Somit startet der evolutionäre Algorithmus mit einer Anfangspopulation, in der sich nur lokale Optima befinden. Dies kann zu einem schnelleren und besseren Verlauf des Algorithmus führen.

4.3.6. Mutation

Peter Kissmann

Die Klasse *Mutation* dient als Schnittstelle für alle Mutations-Operatoren, die in Abschnitt 4.1.3 beschrieben wurden. Hier werden die öffentlichen Methoden, über die alle

abgeleiteten Klassen verfügen, deklariert, aber noch nicht implementiert. Abgeleitet wurden Klassen für die k-Bit-Mutation, die Standardbit-Mutation sowie die Mutation durch Addition eines Mutationsvektors.

4.3.6.1. GAMutationBit

Die Klasse *GAMutationBit* verwirklicht die k-Bit-Mutation. Im Konstruktor werden der Parameter k sowie die Wahrscheinlichkeit, daß in einem Individuum diese k Bits gekippt werden, übergeben. Ebenfalls muß noch ein Parameter übergeben werden, der angibt, ob Individuen, die außerhalb des durch die Nebenbedingungen spezifizierten Suchbereiches liegen, beibehalten und nur als ungültig markiert werden, oder ob sie durch Elternselektion, Rekombination und erneute Mutation komplett neu erzeugt werden.

4.3.6.2. GAMutationStandardBit

Die Klasse *GAMutationStandardBit* implementiert die Standardbit-Mutation. Es muß sowohl die Wahrscheinlichkeit, mit der ein Bit gekippt wird, als auch der oben beschriebene Parameter, der entscheidet, ob ungültige Individuen überleben oder neu erzeugt werden sollen, übergeben werden.

4.3.6.3. ESMutationAdd

Die Klasse *ESMutationAdd* realisiert die Mutation durch Addition eines Mutationsvektors. Sie verfügt über insgesamt vier Konstruktoren, die vor allem den Unterschied zwischen kumulativer Mutationsstärkenadaption und Selbstadaption der Mutationsstärken bilden.

Drei der Konstruktoren dienen der Erzeugung dieser Klasse mit kumulativer Adaption, die in einer eigenen Klasse verwirklicht wird (siehe Abschnitt 4.3.7). Hierzu werden entweder nur die Parameter *memory*, der bestimmt, wie weit die Erinnerung zurückreichen soll, und *dampingFactor* übergeben, der den Dämpfungsfaktor für die Berechnung der neuen Mutationsstärken angibt. In diesem Fall werden die initialen Mutationsstärken zufällig festgelegt, indem eine uniform zufällig verteilte Zufallszahl aus dem Bereich $\left[0, \frac{Cons_i}{4}\right)$ mit $Cons_i$ = Breite des Intervalls des durch die Nebenbedingungen bestimmten Suchraumes in Richtung i. Alternativ können zusätzlich noch zwei reellwertige Felder *left* und *right* übergeben werden, die den initialen Bereich für die Koordinaten einschränken. Dann werden die initialen Mutationsstärken ebenfalls zufällig bestimmt, nun aber in dem Intervall $\left[0, \frac{right-left}{4}\right)$. Eine letzte Alternative bildet die Möglichkeit, statt der linken und rechten Grenzen, die gewünschten Mutationsstärken direkt zu übergeben.

Der vierte Konstruktor dient der Erzeugung dieser Klasse mit Nutzung der Selbstadaption der Mutationsstärken. Hier müssen zwei Parameter *mutationStrengthAdaptationFactor1* sowie *mutationStrengthAdaptationFactor2* mit übergeben werden, die den Parametern τ_1 beziehungsweise τ_2 für die Neuberechnung der Mutationsstärken innerhalb der Mutation entsprechen.

Allen vier Konstruktoren muß noch der Parameter *rotationAngleAdaptationFactor*, der dem Faktor β bei der Neuberechnung der Rotationswinkel im Falle von korrelierter Mutation entspricht, übergeben werden.

Als nächster und mit wichtigster Parameter wird *adaptation* übergeben. Dieser bestimmt, welche Art der Adaption verwendet werden soll:

0 bedeutet, daß keine Adaption verwendet werden soll; in diesem Fall werden globale Mutationsstärken erzeugt, die für alle Individuen dieselben sind, und sich im Laufe des Algorithmus nicht ändern.

2 enspricht der einfachen kumulativen Mutationsstärkenadaption.

4 erzeugt ebenfalls globale Mutationsstärken, die nicht verändert werden. Allerdings werden hierbei auch die Rotationswinkel berücksichtigt, die nur innerhalb der Rekombination angepaßt werden.

5 läuft quasi genauso ab, nur daß hierbei die Rotationswinkel auch innerhalb der Mutation angepaßt werden.

6 entspricht dem Fall 4, außer daß die globalen Mutationsstärken noch der kumulativen Adaption unterzogen werden.

7 schließlich ist der letzte Fall, in dem globale Mutationsstärken verwendet werden. Auch hier werden sie mittels kumulativer Adaption angepaßt. Die Rotationswinkel werden ebenfalls angepaßt, in diesem Fall jedoch innerhalb der Rekombination und der Mutation.

8 ist der erste Fall mit Selbstadaption der Mutationsstärken: Sie werden nur innerhalb der Rekombination angepaßt.

10 fügt noch die Selbstadaption innerhalb der Mutation hinzu.

12 bedeutet, daß die Mutationsstärken und die Rotationswinkel innerhalb der Rekombination angepaßt werden.

13 legt fest, daß die Rotationswinkel auch innerhalb der Mutation mit angepaßt werden.

14 ist die Adaption der Mutationsstärken innerhalb der Rekombination und der Mutation sowie die Anpassung der Rotationswinkel innerhalb der Rekombination.

15 schließlich läßt sowohl die Mutationsstärken als auch die Rotationswinkel in der Rekombination und der Mutation anpassen.

Nur wenn die Rotationswinkel angepaßt werden, werden sie auch für die Berechnung der Mutationsvektoren benutzt. Anderenfalls wird bei dieser Berechnung auf sie verzichtet und man erhält eine unkorrelierte Mutation.

Wie in den beiden zuvor beschriebenen Verfahren wird in allen Fällen noch ein letzter Parameter benötigt, der angibt, wie mit ungültige Individuen zu verfahren ist. Auch hier gibt es nur die Wahl zwischen beibehalten und als ungültig markieren sowie verwerfen und neu erzeugen.

4.3.7. MutationStrengthAdaptation

Peter Kissmann

Die Klasse *MutationStrengthAdaptation* wird nur für die kumulative Mutationsstärkenadaption benötigt. Sie speichert zum einen die globalen Mutationsstärken, die bei der

Initialisierung im Konstruktor übergeben werden müssen, zum anderen den Fortschrittsvektor, der für die Berechnung der neuen Mutationsstärken nötig ist und stets am Ende eines Zyklus (nach der Umweltselektion) aktualisiert wird.

Diese Klasse wird nur zusammen mit *ESAddMutation* benötigt, da in keinem anderen Operator die Mutationsstärken verwendet werden. Daher wird diese Klasse im Gegensatz zu den restlichen auch nicht global vom *Controller* erzeugt, sondern innerhalb des Konstruktors der *ESAddMutation*. Für den Konstruktor müssen die Mutationsstärken (die notfalls während der Erzeugung der *ESAddMutation* berechnet werden) sowie die Parameter *memory* und *dampingFactor* übergeben werden.

Die Adaption erfolgt dann gemäß der Beschreibung innerhalb der theoretischen Grundlagen und muß stets nach der Umweltselektion aufgerufen werden.

4.3.8. Population

Peter Kissmann

Die Klasse *Population* ermöglicht das Erstellen und Verwalten einer Population von Individuen. Dem Konstruktor müssen hierzu die Populationsgröße sowie die erzeugten Individuen (als Feld) übergeben werden. Objekte dieser Klasse sind dann in der Lage, einzelne Individuen oder die Konfigurationen, die den Individuen zugeordnet sind zurückzugeben, sowie die gesamte Population oder nur einen Teilbereich gemäß der Fitneß-Schätzung aufsteigend zu sortieren.

4.3.9. Recombination

Christian Horoba

Die abstrakte Klasse *Recombination* ist die Oberklasse aller Klassen, die konkrete Rekombinationsoperatoren implementieren. Sie dient als gemeinsame Schnittstelle aller Rekombinationsoperatoren. Die beiden zentralen Methoden sind *virtual void recombine()* und *virtual void recombine(Individual& rChild)*. Die erstgenannte Methode ist bereits vollständig implementiert und füllt im Wesentlichen die Nachkommenpopulation mit neu generierten Individuen. Die Konstruktion dieser Individuen erfolgt jeweils durch einen Aufruf der zweitgenannten Methode. Letztere Methode ist abstrakt und muß folglich von jedem konkreten Rekombinationsoperator implementiert werden, um die gewünschte Funktionalität bereitzustellen. Dem Konstruktor der Klasse müssen folgende Argumente übergeben werden: *rAlgorithm*, *rParentSelection*, *rConstraints*, *handleInvalidity*, *recoProb*, *adaptation*, *dimension*.

Die Referenz *rAlgorithm* macht ein Objekt der Klasse *EvolutionaryAlgorithm* bekannt. Die Referenz *rParentSelection* verweist auf ein Objekt einer Unterklasse der abstrakten Klasse *Selection*, das die gewünschte Art der Elternselektion realisiert. Die Referenz *rConstraints* verweist auf ein Objekt der Klasse *Constraints*, das Informationen über die Nebenbedingungen enthält. Die natürliche Zahl *handleInvalidity* beschreibt, wie mit ungültigen Nachkommen, die unter Umständen durch den Rekombinationsprozeß entstehen können, zu verfahren ist. Die Gleitkommazahl *recoProb* definiert die Wahrscheinlichkeit, mit der der Rekombinationsoperator angewandt wird. Wenn keine Individuen

rekombiniert werden sollen, wird das erste selektierte Elterindividuum in die Nachkommenpopulation übernommen. Die natürliche Zahl *adaptation* regelt, ob nur die Koordinaten oder auch die Mutationsschrittweiten oder Rotationswinkel dem Rekombinationsprozeß unterworfen werden. Die natürliche Zahl *dimension* macht die Dimension des Suchraums bekannt.

4.3.9.1. RecoWithout

Diese Klasse implementiert keinen Rekombinationsoperator im eigentlichen Sinne. Sie dient gewissermaßen dazu, den Rekombinationsprozeß zu deaktivieren. Konkret bedeutet das, daß sie ein selektiertes Elterindividuum unverändert in die Nachkommenpopulation überträgt. Dem Konstruktor der Klasse müssen folgende Argumente übergeben werden: *rAlgorithm*, *rParentSelection*, *rConstraints*, *dimension*.

Die Bedeutung dieser Argumente entspricht der im Rahmen der Oberklasse *Recombination* beschriebenen. Bei der Benutzung dieser Klasse ist zu beachten, daß in der Konfigurationsdatei der Wert des Parameters PNUMBEROFPARENTS auf 1 gesetzt werden muß.

4.3.9.2. GARecoCross

Diese Klasse implementiert den k-Punkt-Crossover-Rekombinationsoperator und einige seiner Varianten. Die folgenden Parameter dienen seiner Konfiguration: *rAlgorithm*, *rParentSelection*, *rConstraints*, *handleInvalidity*, *recoProb*, *adaptation*, *dimension*, *bitsPerCoordinate*, *numberOfCuts*, *numberOfChildren*, *onlyCutBetweenCoordinates*, *shuffle*.

Die natürliche Zahl *bitsPerCoordinate* enthält die Anzahl Bits, mit denen eine Koordinate repräsentiert wird. Der Parameter *numberOfCuts* definiert die Anzahl der Schnitte, mit denen die Bitstrings geteilt werden. Der Parameter *numberOfChildren* dient der Einstellung der Anzahl gewünschter Nachkommen. Dieser Wert muß zwischen 1 und dem Wert von PNUMBEROFPARENTS liegen. Der Parameter *onlyCutBetweenCoordinates* gibt an, ob die Schnitte an beliebigen Positionen oder nur zwischen den Bitrepräsentationen zweier Koordinaten gemacht werden dürfen. Der Parameter *shuffle* enthält die Information, ob die Bits vor der Anwendung des eigentlichen Rekombinationsoperators beliebig permutiert werden sollen. Nach der Rekombination werden die Bits dann durch die Anwendung der inversen Permutation wieder in ihre ursprüngliche Reihenfolge gebracht.

4.3.9.3. GARecoUniCross

Diese Klasse implementiert den uniformen Crossover-Rekombinationsoperator. Dieser Operator zeichnet sich dadurch aus, daß für jede Position unabhängig uniform zufällig entschieden wird, von welchem Elterindividuum die entsprechende Information übernommen wird. Die folgenden Argumente müssen dem Konstruktor übergeben werden: *rAlgorithm*, *rParentSelection*, *rConstraints*, *handleInvalidity*, *recoProb*, *adaptation*, *dimension*, *bitsPerCoordinate*.

4.3.9.4. ESReco2Intermediate

Diese Klasse implementiert verschiedene Formen der intermediären Rekombination für genau zwei Elterindividuen. Der genaue Ablauf kann mit den folgenden Parametern festgelegt werden: *rAlgorithm*, *rParentSelection*, *rConstraints*, *handleInvalidity*, *recoProb*, *adaptation*, *dimension*, *halfWidth*, *line*.

Die Parameter *halfWidth* und *line* bestimmen den Bereich, aus dem uniform zufällig ein Nachkomme ausgewählt wird. Hier ist zu beachten, daß der Wert des Parameters PNUMBEROFPARENTS genau 2 betragen muß.

4.3.9.5. ESRecoDiscrete

Diese Klasse realisiert verschiedene Formen der diskreten Rekombination. Das Verhalten läßt sich mit den folgenden Parametern beeinflussen: *rAlgorithm*, *rParentSelection*, *rConstraints*, *handleInvalidity*, *recoProb*, *adaptation*, *dimension*, *holdFirstParent*.

Wenn der Parameter *holdFirstParent* auf 0 gesetzt wird, wird für jede Komponente unabhängig uniform zufällig entschieden, von welchem der ausgewählten Elterindividuen sie übernommen wird. Wenn der Parameter *holdFirstParent* auf 1 gesetzt wird, wird das erste Elterindividuum festgehalten und für jede Komponente ein weiteres Individuum ausgewählt. Anschließend wird jeweils entschieden, von welchem der beiden Individuen die entsprechende Information übernommen wird. In letzterem Fall ist zu beachten, daß der Wert des Parameters PNUMBEROFPARENTS dem durch die Dimension und die Art der Adaption bestimmten entsprechen muß. Beispielsweise muß der Parameter PNUMBEROFPARENTS den Wert $1+2\cdot\text{DIMENSION}+(\text{DIMENSION}-1)\cdot\text{DIMENSION}/2 = 10$ enthalten, wenn DIMENSION = 3 und ADAPTATION = 15 gilt.

4.3.9.6. ESRecoIntermediate

Diese Klasse implementiert genau wie die Klasse *ESReco2Intermediate* eine Form der intermediären Rekombination. Hier sind jedoch auch mehr als zwei Elterindividuen erlaubt. Die Komponenten des Nachkommens ergeben sich hierbei als Mittelwerte der entsprechenden Komponenten der Elterindividuen. Die folgenden Parameter dienen der Konfiguration des Operators: *rAlgorithm*, *rParentSelection*, *rConstraints*, *handleInvalidity*, *recoProb*, *adaptation*, *dimension*, *holdFirstParent*.

Die Einstellung des Parameters *holdFirstParent* erfolgt auf dieselbe Art und Weise wie für die Klasse *ESRecoDiscrete*.

4.3.9.7. ESRecoSchwefel

Diese Klasse implementiert eine Mischform aus diskreter und intermediärer Rekombination. Die Koordinaten entstehen durch paarweise diskrete Rekombination, während die Schrittweiten und Rotationswinkel gegebenenfalls durch globale intermediäre Rekombination gewonnen werden. Die folgenden Parameter müssen angegeben werden: *rAlgorithm*, *rParentSelection*, *rConstraints*, *handleInvalidity*, *recoProb*, *adaptation*, *dimension*.

Hierbei ist zu beachten, daß der Wert des Parameters PNUMBEROFPARENTS dem durch die Dimension und die Art der Adaption bestimmten entsprechen muß.

4.3.10. Selection

Katharina Balzer, Peter Kissmann

Die Klasse *Selection* dient als Oberklasse für die verschiedenen Selektionsmethoden, d. h. sie verwaltet als Container diejenigen Attribute, die alle Selektionsoperatoren gemeinsam haben. Die Selektion im allgemeinen dient zum einen dazu, Eltern für den Rekombinationsprozeß auszuwählen (Elternselektion) und zum anderen sollen nach jeder Generation Individuen für die neue Elternpopulation ausgewählt werden (Umweltselektion). Hierbei unterscheidet man, ob man nur aus der Kindpopulation (Komma-Selektion) oder aus der Kind- und der alten Elternpopulation (Plus-Selektion) auswählt.
Die hier implementierten Selektionsoperatoren sind *SelectUniform*, *SelectDeterministic*, *SelectTournament* und *SelectFitnessProp*. In allen Fällen können die Individuen mit oder ohne Zurücklegen gezogen werden. Bei der Umweltselektion kann man zwischen den eben erwähnten Plus- und Komma-Selektionen wählen. Für alle diese Wahlmöglichkeiten stehen boolesche Parameter zur Verfügung, die je nach Konstellation einen der Fälle ergeben.
Desweiteren gilt im Falle der Elternselektion, daß man, wenn man ohne Zurücklegen zieht, nicht mehr Indiviuen ziehen kann, als in der Auswahlpopulation enthalten sind. Wird dies versucht, so erhält man eine Fehlermeldung, bevor das Programm startet. Außerdem wird die Elternselektion direkt von der Rekombination aus gestartet, während die Umweltselektion vom Algorithmus angestoßen wird.

4.3.10.1. SelectUniform

Bei der uniformen Selektion wird eine ganzzahlige Zufallszahl gezogen und dann dasjenige Individuum ausgewählt, das an der Indexposition in der Population steht, welche der Zufallszahl entspricht. Jedes der gewählten Individuen wird übernommen. Befindet man sich in einer Elternselektion, so werden so viele Individuen gezogen, wie Eltern für eine Rekombination nötig sind, befindet man sich in einer Umweltselektion, so werden so viele Individuen gezogen, wie die Elternpopulation groß ist. Bei der Umweltselektion werden anschließend die gewählten und zwischengespeicherten Individuen in die alte Elternpopulation kopiert und somit die alte Elternpopulation überschrieben.

4.3.10.2. SelectDeterministic

Die deterministische Selektion hängt stark von den Fitneßwerten der Individuen ab, da immer die Individuen mit der besten Fitneß deterministisch gewählt werden. Um diese Individuen auszuwählen, findet hier ein Ranking & Selection-Verfahren Anwendung. Dabei werden alle Individuen - eventuell nach zusätzlichen Auswertungen - verglichen und, abhängig vom jeweiligen Verfahren, entschieden, welche tatsächlich die besten darstellen. Bei der Elternselektion werden dann diese ausgewählten besten Individuen an die Rekombination weitergereicht, bei der Umweltselektion bilden sie die neue Elternpopulation.

Da es hier keinen Sinn machen würde, das gleiche Individuum mehrfach auszuwählen, wird der Fall, daß Individuen doppelt gezogen werden können, ausgeschlossen.

4.3.10.3. SelectTournament

Bei der Turnierselektion findet in jeder Runde ein Turnier mehrerer Individuen statt. Wieviele Individuen in ein Turnier kommen, hängt von der Turniergröße ab, die als Parameter im Konstruktor mit übergeben wird. Sie wird vom Benutzer festgelegt. Aus den Individuen in jedem Turnier wird dann jeweils das mit der besten Fitneß ausgewählt. Auch hier findet ein Ranking & Selection-Verfahren Anwendung, das für die einzelnen Individuen - eventuell durch zusätzliche Auswertungen - die jeweilige Fitneß bestimmt.

4.3.10.4. SelectFitnessProp

Die fitneßproportionale Selektion ist die komplizierteste der vier vorgestellten Möglichkeiten, Individuen zu selektieren. Hier werden die Individuen der Auswahlpopulation erst in ein lineares Ranking gebracht, welches mit Hilfe eines bestimmten Selektionsdrucks (in unserem Fall meistens 2) berechnet wird. Um dies möglich zu machen, werden die Individuen zunächst durch ein Ranking & Selection-Verfahren und damit verbundene eventuelle Auswertungen anhand ihrer Fitneß aufsteigend sortiert. Gemäß des berechneten Rankings werden den Individuen dann neue Fitneßwerte zugewiesen. Bei einem Minimierungsproblem bekommt dann das Individuum mit der kleinsten Rankingposition die größte Fitneß zugewiesen, bei Maximierungsproblemen genau umgekehrt. Mittels der neuen Fitneßwerte wird eine Verteilung berechnet, gemäß der die Individuen nun gezogen werden. Hier ist klar, daß es nun wahrscheinlicher ist, ein Individuum zu ziehen, welches eine gute Fitneß besitzt, also dem Optimum recht nah kommt.

4.3.11. TerminationCriterion

Christian Horoba

Diese abstrakte Klasse ist die Oberklasse aller Abbruchbedingungen und deklariert die abstrakte Methode *virtual bool terminate()*. Diese Methode wird von konkreten Subklassen implementiert, um ein spezifisches Abbruchkriterium zu realisieren. Vor jedem Schleifendurchlauf des Evolutionszyklus wird die genannte Methode aufgerufen und genau dann, wenn diese den Wahrheitswert *true* liefert, abgebrochen. Dem Konstruktor dieser Klasse muß nur durch den Parameter *rAlgorithm* das Objekt der Klasse *EvolutionaryAlgorithm* bekannt gemacht werden.

4.3.11.1. TermChangeless

Diese Klasse bricht die Suche nach einer optimalen Lösung ab, wenn über mehrere aufeinander folgende Generationen kein ausreichender Fortschritt mehr erzielt wurde. Die folgenden Parameter dienen der Konfiguration dieses Abbruchkriteriums: *rAlgorithm*, *gens*, *max*, *threshold*.

Der Wert des Parameters *gens* bestimmt die Anzahl der oben erwähnten Generationen; der des Parameters *max* gibt an, ob es sich um ein Minimierungs- oder Maximierungsproblem handelt. Der Wert des Parameters *threshold* definiert einen Schwellwert. Ein positiver Wert veranlaßt das Abbruchkriterium marginale Verbesserungen nicht als solche zu betrachten und den Optimierungsalgorithmus gegebenenfalls abzubrechen, obwohl tatsächlich noch geringe Verbesserungen erzielt wurden.

4.3.11.2. TermEvals

Dieses Abbruchkriterium bricht nach der Durchführung einer bestimmten Anzahl von Funktionsauswertungen ab. Die folgenden Parameter müssen dem Konstruktor übergeben werden: *rAlgorithm*, *noOfEvals*.

Der Wert des Parameters *noOfEvals* definiert die oben genannte Anzahl von Funktionsauswertungen. Dabei ist zu beachten, daß die Möglichkeit eines Abbruchs des Optimierungsverfahrens nur unmittelbar vor dem Beginn eines erneuten Schleifendurchlaufs des Evolutionszyklus besteht. Daher kann es dazu kommen, daß tatsächlich mehr als die hier eingestellte Anzahl von Funktionsauswertungen durchgeführt werden.

4.3.11.3. TermGeneration

Dieses Abbruchkriterium beendet den Algorithmus nach einer festen Anzahl berechneter Generationen. Die folgenden Parameter müssen dem Konstruktor übergeben werden: *rAlgorithm*, *gens*.

Der Parameter *gens* dient der Spezifikation der oben genannten Anzahl Generationen.

4.3.11.4. TermThreshold

Diese Abbruchbedingung bricht genau dann ab, wenn die Fitneß des besten gefundenen Individuums nicht größer bzw. nicht kleiner als der übergebene Wert ist. Die folgenden Parameter müssen dem Konstruktor übergeben werden: *rAlgorithm*, *threshold*, *max*.

Der Parameter *threshold* dient der Übergabe des entsprechenden Schwellwertes und der Parameter *max* gibt an, ob es sich um ein Minimierungs- oder Maximierungsproblem handelt.

4.4. Leistungsstudien

4.4.1. Einleitung

Christian Horoba

Dieser Abschnitt stellt die Ergebnisse der Leistungsstudien dar. Es wurde die Performanz verschiedener evolutionärer Algorithmen untersucht.

An die Stelle der Fitneßfunktion traten die neun vorgestellten theoretischen Testfunktionen. Die folgenden Dimensionen des Suchraums wurden untersucht: 2, 4, 6, 8, 10. Hierbei ist zu beachten, daß Schaffers Funktion nur für die Dimension 2 definiert ist.

Konkret wurde der Suchraum auf die Menge $[-2, 2]^n$ begrenzt, wobei n die Dimension des Suchraums bezeichnet.

Ursprünglich war noch die Untersuchung von drei ausgewählten APNN-Simulationsmodellen (Ampelkreuzung, Fertigungssystem, Tankstelle) geplant. Diese mußte jedoch aufgrund technischer Schwierigkeiten mit dem entsprechenden Simulator vorerst ausfallen.

Es wurden vier verschiedene Sorten von Diagrammen konstruiert, um jeweils bestimmte Eigenschaften der Algorithmen zu beleuchten. Die erste Sorte stellt auf der x-Achse die Anzahl der Funktionsauswertungen und auf der y-Achse die Fitneß der besten bisher gefundenen Lösung als Prozentsatz bezüglich der Größe des Wertebereichs dar. Dabei entspricht der maximale Funktionswert 0 Prozent und der minimale Funktionswert 100 Prozent. Diese Grafik veranschaulicht mit einer Kurve den Ablauf einer einzelnen Suche nach einer optimalen Lösung. Da für die Simulationsmodelle a priori nicht bekannt ist, welche Werte auf einem bestimmten Definitionsbereich angenommen werden, wurde in diesem Fall die Fitneß der besten bisher gefundenen Lösung als absoluter Wert abgetragen. Es läßt sich erkennen, mit welchem Aufwand eine Lösung, die eine bestimmte Güte bezüglich der Fitneßfunktion erfüllt, gefunden werden kann. Nur anhand dieser Grafik läßt sich jedoch nicht beurteilen, ob der Algorithmus eine Lösung, deren Abstand zu einer optimalen Lösung klein ist, oder „nur“ eine Lösung, die trotz eines kleinen Fitneßwerts weit von einer optimalen Lösung entfernt liegt, gefunden hat. Die zweite Sorte stellt auf der x-Achse ebenfalls die Anzahl der Funktionsauswertungen und auf der y-Achse den euklidischen Abstand der besten bisher gefundenen Lösung zu einer optimalen Lösung dar. Diese Grafiken füllen die oben geschilderte Lücke. Die genannten Diagramme konnten aus nahe liegenden Gründen nur für die theoretischen Benchmarkfunktionen erstellt werden.

Des Weiteren wurden die Auswirkungen verrauschter Funktionsauswertungen untersucht. Um verrauschte Funktionsauswertungen zu realisieren, wurde zu dem tatsächlichen Funktionswert die Realisation einer normalverteilten Zufallsvariable mit Erwartungswert 0 und an die Fitneßfunktion angepaßter – und von der Stärke des Rauschens abhängiger – Standardabweichung addiert. Die Standardabweichung wurde auf einen festen Anteil des maximalen Funktionswerts auf dem betrachteten Definitionsbereich gesetzt. Die folgenden Stärken wurden untersucht: 0.00, 0.04, 0.08, 0.12, 0.16, 0.20. Man beachte, daß also fitneßunabhängiges gaußsches Rauschen verwendet wurde.

Die dritte Diagrammsorte stellt auf der x-Achse die Stärke des Rauschens und auf der y-Achse die tatsächliche Fitneß der besten bisher gefundenen Lösung als Prozentsatz des bekannten Optimums dar. Die Stärke des Rauschens wird gemäß den obigen Erklärungen als Prozentsatz abgetragen. Die vierte Sorte stellt auf der x-Achse ebenfalls die Stärke des Rauschens und auf der y-Achse die Anzahl benötigter Funktionsauswertungen dar. Diese Anzahl ist natürlich von dem gewählten Abbruchkriterium des Algorithmus abhängig. Die beiden letztgenannten Grafiken konnten nur für die Benchmarkfunktionen erstellt werden, da die Stärke des Rauschens im Falle der Simulationsmodelle nicht direkt beeinflußt werden konnte. Anhand der genannten Grafiken läßt sich beurteilen, wie robust der jeweilige Algorithmus auf das Hinzufügen von Rauschen reagiert. D. h. konkret, wie die Güte der gefundenen Lösung zu beurteilen ist und wie viele Funktionsauswertungen

für ihre Bestimmung nötig waren.

Da das Konzept der evolutionären Algorithmen als „Baukasten“ zu verstehen ist, ergibt sich eine sehr große Vielfalt verschiedener Algorithmen. Eine erschöpfende Analyse aller denkbaren Konfigurationen ist also nicht praktikabel. Um trotzdem möglichst viele verschiedene Konfigurationen zu testen, wurde folgendermaßen vorgegangen: Es wurden zwei Grundkonfigurationen erstellt und anschließend die Auswirkungen des Austauschs einzelner Komponenten untersucht. Die beiden Grundkonfigurationen definieren einen klassischen genetische Algorithmus und eine Evolutionsstrategie. Beide Algorithmen sind typische Instanzen evolutionärer Algorithmen.

An dieser Stelle wird nur grob auf die Konfigurationen der verwendeten Algorithmen eingegangen. Details sind den beiden im Anhang abgedruckten Konfigurationsdateien zu entnehmen. Die Größe der Elternpopulation wurde auf 10 und die Größe der Nachkommenpopulation auf 20 gesetzt. Die Initialisierung der Elternpopulation erfolgte unabhängig uniform zufällig. Die Selektion der Eltern erfolgte im Falle des genetischen Algorithmus fitneßproportional und im Falle der Evolutionsstrategie uniform zufällig. Die Rekombination erfolgte durch 1-Punkt-Crossover bzw. eine Mischung aus diskreter und intermediärer Rekombination. Die Selektion der nächsten Population erfolgte in beiden Fällen deterministisch, wobei sowohl aus der aktuellen Eltern- als auch Nachkommenpopulation gewählt werden durfte (Plus-Selektion). Der Algorithmus wurde in beiden Fällen für die Konstruktion der ersten beiden Diagramme nach 2000 Funktionsauswertungen abgebrochen. Für die Erstellung der letzten beiden Grafiken wurde der Algorithmus abgebrochen, sobald über 5 aufeinander folgende Generationen keine nennenswerte Verbesserung mehr erzielt wurde. Der entsprechende Schwellwert wurde hier auf das 0.01-fache des jeweiligen maximalen Funktionswerts gesetzt.

Teilweise beruhen die erstellten Diagramme auf 20 Replikationen. Die Diagramme, die den Einfluß der Populationsgrößen, der Selektionsoperatoren und der Abbruchbedingung darstellen, basieren auf einer Replikation. Durch die anschließende Mittelwertbildung soll sichergestellt werden, daß ein typischer Optimierungsverlauf dargestellt wird. Aufgrund technischer Schwierigkeiten konnte die Unabhängigkeit der verwendeten Zufallszahlensequenzen jedoch nicht sichergestellt werden.

Im Detail werden folgende Studien vorgestellt:

1. Einfluß der Populationsgrößen
2. Einfluß der Selektionsoperatoren
3. Einfluß des Rekombinationsoperators
4. Einfluß des Mutationsoperators
5. Einfluß des Abbruchkriteriums
6. Einfluß des Auswahlverfahrens für die Verwaltung der Elitepopulation

4.4.2. Populationsgrößen

Julia Hielscher

Im folgenden Abschnitt wird der Einfluß der Populationsgröße bei Komma- und Plus-Selektion untersucht. Die veränderten Parameter waren also:

- die Größe der Elternpopulation, die in dem Bereich 1 bis 50 untersucht wurde
- die Größe der Kindpopulation, die ebenfalls in dem Bereich 1 bis 50 untersucht wurde und
- der Parameter EPLUSSELECTION, welcher bei Komma-Selektion 0 und bei Plus-Selektion 1 ist.

Bei der Durchführung der Leistungsstudien wurden die Kombinationen zudem so eingeschränkt, daß die Größe der Kindpopulation stets größer oder gleich der Größe der Elternpopulation war.
Für die Untersuchungen ohne Rauschen ergab sich:
Sowohl für die verwendete Evolutionsstrategie als auch für den genetischen Algorithmus ergab sich, daß sehr kleine Populationsgrößen von 1 oder 5 nicht geeignet sind. Hier sind viele Auswertungen notwendig um eine Verbesserung zu erzielen und auch dann wenn Verbesserungen erzielt werden, so liegen die Ergebnisse immer noch in einem schlechten Bereich. Zudem zeigte sich, daß die Wahl gleicher Populationsgrößen für die beiden Populationen nicht wirkungsvoll ist. Für eine Komma-Strategie war dies zu erwarten, da ja dann stets alle Individuen in die nächste Generation übernommen werden. Aber auch die Plus-Selektion liefert bei dieser Einstellung schlechtere Ergebnisse als bei anderen Einstellungen. Verdeutlicht wird dies in Abbildung 4.3. Dort ist für die 2-dimensionale Ackley Funktion und Populationsgröße 20 zu sehen, daß bei der Komma-Selektion nur eine 65.4% gute Lösung erzielt wird und daß während der Optimierung keine Verbesserungen stattfinden. Die Plus-Selektion bei gleichen Eingaben dagegen findet das Optimum, allerdings erst nach 1400 Auswertungen.

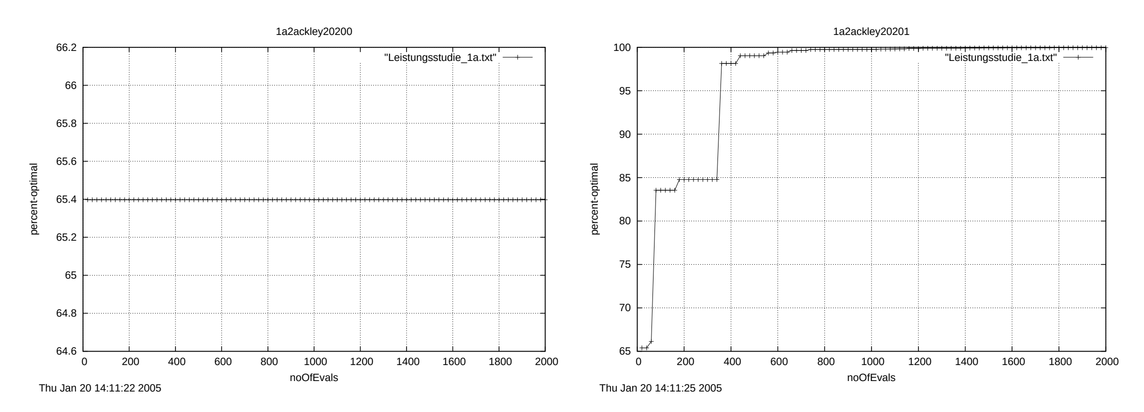


Abb. 4.3.: Komma- (links) und Plus-Selektion bei gleichen Populationsgrößen und 2-dimensionaler Ackley Funktion

Insgesamt gesehen hat sich die Plus-Selektion bei allen verwendeten Funktionen als besser heraugestellt. Jedoch ist bei Evolutionsstrategien in Verbindung mit der Komma-Selektion zu beobachten, daß sobald die Popualtionsgrößen in einem Verhältnis Elternpopulationsgröße / Kindpopulationsgröße von 1/7 bis 1/5 stehen, daß dann die erzielten Ergebnisse auch hier gut sind. Die Vorteilhaftigkeit dieses Verhältnisses zeigt sich auch in Abbildung 4.4. In letzterer ist links zu erkennen, daß bei der 4-dimensionalen Sphere Funktion und Populationsgrößen von 20 und 30, also einem Verhältnis von 2/3 ein für die Komma-Selektion typisches Bild entsteht: wenige Verbesserungsschritte und ein Ergebnis von 95.75%. Im rechten Teil der Abbildung jedoch stehen die Populationsgrößen in einem Verhältnis von 1/5. Hier sind deutlich mehr Verbesserungsschritte erkennbar und auch das Endergebnis von 99.4% ist besser.

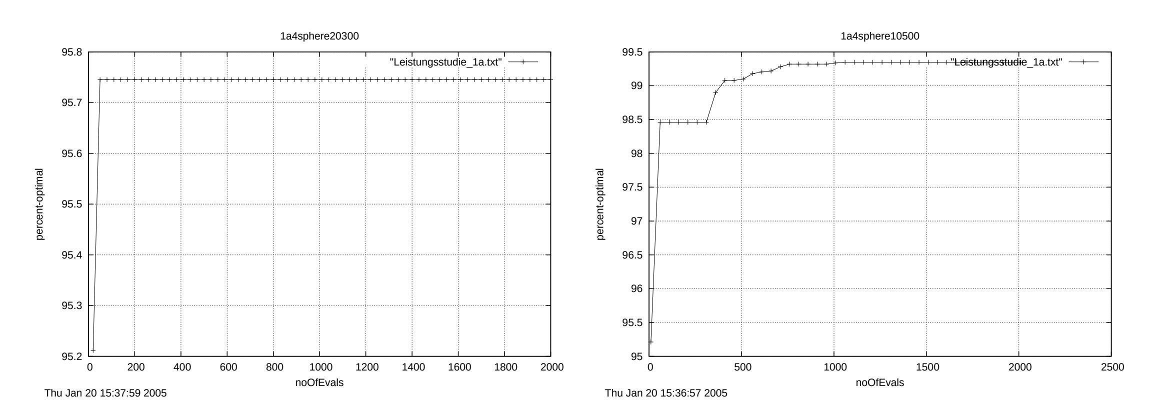


Abb. 4.4.: Komma-Selektion bei Populationsgrößen 20 und 30 (links) bzw. 10 und 50 (rechts) bei 4-dimensionaler Sphere Funktion

4.4.3. Selektion

Katharina Balzer

In den nun im folgenden beschriebenen Leistungsstudien ging es darum, den Einfluß der Selektion, insbesondere der Turniergröße bei der Turnierselektion zu untersuchen. Als Beispielfunktionen dienten Ackleys, Schaffers und die Sphere-Funktion. Die getesteten Turniergrößenwerte waren 2, 4, 6 und 8 bei einer Elternpopulationsgröße von 10. Desweiteren fanden die Leistungsstudien für die Dimensionen 2, 4, 6 und 8 statt. Alle Ergebnisse wurden mit einer Replikation eines Durchlaufs erstellt.

Bei den Untersuchungen ohne Rauschen ergaben sich folgende Ergebnisse:

Bei der Sphere-Funktion stellte sich eine deterministische Umweltselektion als optimal heraus. Das gleiche gilt vermutlich für alle weiteren unimodalen Funktionen. Beispielhafte Kurven für die Sphere-Funktion sind in Abbildung 4.5 dargestellt. Dort kann man erkennen, daß der Algorithmus nach ca. 110 Funktionsauswertungen das Optimum erreicht.

Für die Ackleyfunktion schien eine Turnierselektion als Umweltselektion gute Ergebnisse zu liefern. Dabei erwiesen sich bei einem Durchlauf ohne Zurücklegen die Turnier-

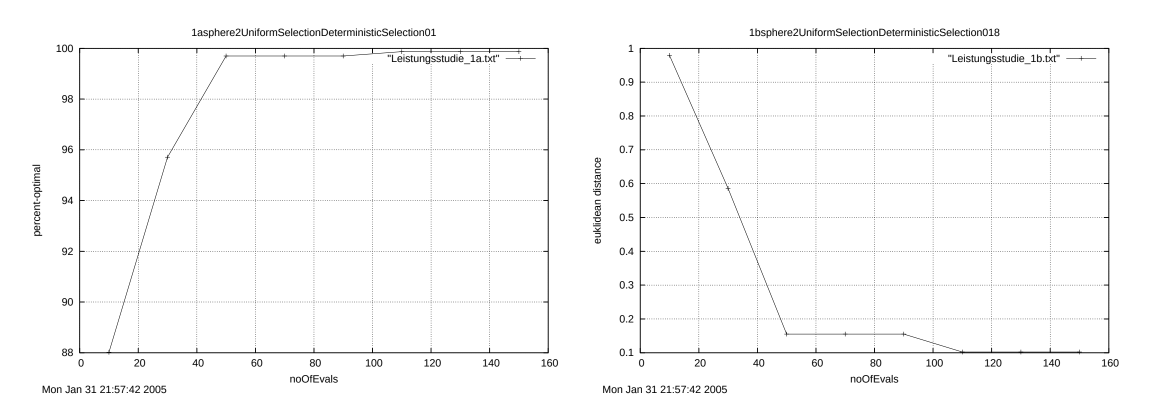


Abb. 4.5.: Deterministische Umweltselektion bei der zweidimensionalen Sphere-Funktion

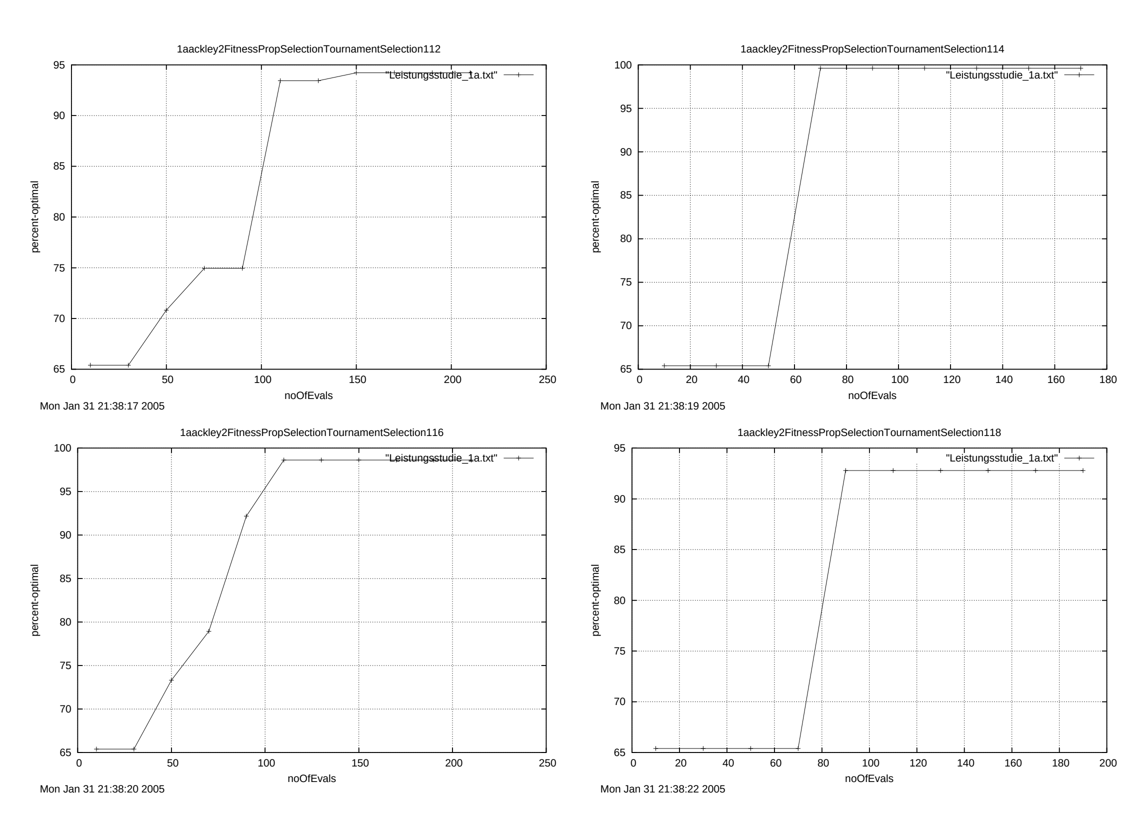


Abb. 4.6.: Ergebnisse mit Turnierselektion bei Ackleys Funktion

größen 4 und 6 als gut, bei Durchläufen mit Zurücklegen die Turniergrößen 6 und 8. Die Turniergröße 2 lieferte in allen Fällen schlechtere Ergebnisse. Als besonders schlecht für die Ackleyfunktion erwies sich eine uniforme Umweltselektion. Ergebnisse sind in den Abbildung 4.6 dargestellt. Dabei sieht man in den vier verschiedenen Graphiken die Ergebnisse für die vier untersuchten Turniergrößen 2, 4, 6 und 8.

Bei der Untersuchung der Schaffer-Funktion stellte sich heraus, daß eine fitneßproportionale Umweltselektion gute Dienste leisten könnte. Ein Ergebnis ist in Abbildung 4.7 zu sehen.

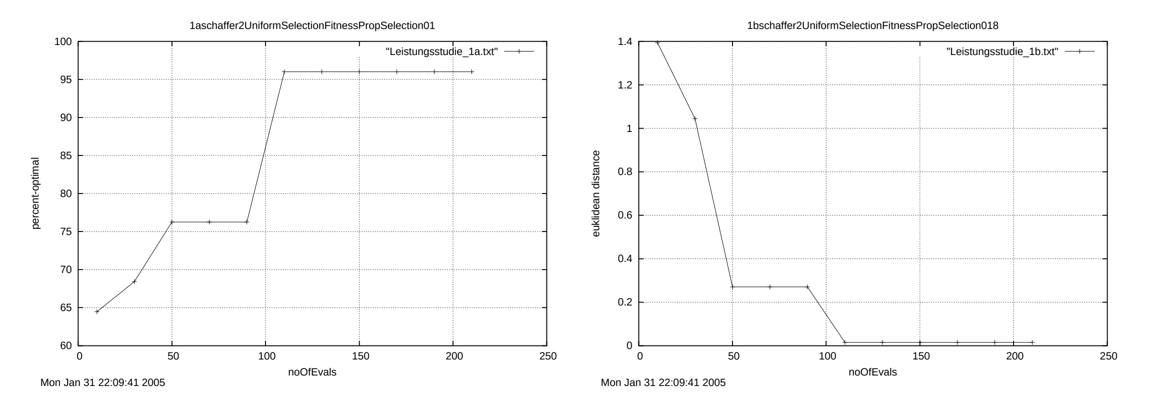


Abb. 4.7.: Ergebnisse für Schaffers Funktion bei fitneßproportionaler Umweltselektion

4.4.4. Rekombinationsoperatoren

Christian Horoba

Dieser Abschnitt stellt einige Ergebnisse der Untersuchung des Einflusses des Rekombinationsoperators auf die Performanz des evolutionären Algorithmus dar.

Es wurden alle implementierten Operatoren untersucht, wobei natürlich nicht alle Variationsmöglichkeiten betrachtet werden konnten. Es wurde eine Einschränkung auf gebräuchliche Konfigurationen der einzelnen Operatoren vorgenommen. Für alle Operatoren wurden folgende Einstellungen festgehalten:

- RECOMBINATIONPROBABILITY = 1.0
- HANDLEINVALIDITY = 0

Die folgenden Parametereinstellungen wurden für die angegebenen Operatoren untersucht:

1. RECOMBINATION = WithoutRecombination
   Basiskonfig. $\in \{\mathrm{GA}, \mathrm{ES}\}$
2. RECOMBINATION = GACrossoverRecombination
   (Basiskonfig., ADAPTATION) $\in \{(\mathrm{GA}, 0), (\mathrm{ES}, 15)\}$
   PNUMBEROFPARENTS $\in \{2, 3\}$

$\text{ONLYCUTBETWEENCOORDINATES} \in \{0, 1\}$
$\text{NUMBEROFCUTS} \in \{1, 2\}$
$\text{NUMBEROFCHILDREN} \in \{1, \text{PNUMBEROFPARENTS}\}$
$\text{SHUFFLE} \in \{0, 1\}$

3. $\text{RECOMBINATION} = \text{GAUniCrossoverRecombination}$
   $(\text{Basiskonfig.}, \text{ADAPTATION}) \in \{(\text{GA}, 0), (\text{ES}, 15)\}$
   $\text{PNUMBEROFPARENTS} \in \{2, 3\}$

4. $\text{RECOMBINATION} = \text{ESReco2Intermediate}$
   $(\text{Basiskonfig.}, \text{ADAPTATION}) \in \{(\text{GA}, 0), (\text{ES}, 15)\}$
   $\text{PNUMBEROFPARENTS} = 2$
   $\text{HALFWIDTH} \in \{0.50, 0.75, 1.00\}$
   $\text{LINE} \in \{0, 1\}$

5. $\text{RECOMBINATION} = \text{ESDiscreteRecombination}$
   $(\text{Basiskonfig.}, \text{ADAPTATION}, \text{PNUMBEROFPARENTS}, \text{HOLDFIRSTPARENT})$
   $\in \{(\text{GA}, 0, 2, 0), (\text{GA}, 0, 1 + \text{DIMENSION}, 1), (\text{ES}, 15, 2, 0)\}$
   $\cup \left\{ \left(\text{ES}, 15, 1 + \text{DIMENSION} + \text{DIMENSION} + \frac{(\text{DIMENSION}-1) \cdot \text{DIMENSION}}{2}, 1 \right) \right\}$

6. $\text{RECOMBINATION} = \text{ESIntermediateRecombination}$
   $(\text{Basiskonfig.}, \text{ADAPTATION}, \text{PNUMBEROFPARENTS}, \text{HOLDFIRSTPARENT})$
   $\in \{(\text{GA}, 0, 2, 0), (\text{GA}, 0, 1 + \text{DIMENSION}, 1), (\text{ES}, 15, 2, 0)\}$
   $\cup \left\{ \left(\text{ES}, 15, 1 + \text{DIMENSION} + \text{DIMENSION} + \frac{(\text{DIMENSION}-1) \cdot \text{DIMENSION}}{2}, 1 \right) \right\}$

7. $\text{RECOMBINATION} = \text{ESSchwefelRecombination}$
   $\text{Basiskonfig.} = \text{ES}$
   $\text{ADAPTATION} = 15$
   $\text{PNUMBEROFPARENTS} = 1 + 1 + \text{DIMENSION} + \frac{(\text{DIMENSION}-1) \cdot \text{DIMENSION}}{2}$

Die Anzahl der Funktionsauswertungen einer Lösung wurde für die Erstellung der ersten bzw. letzten beiden Diagramme fest auf 1 bzw. 20 eingestellt.

Im Folgenden werden zunächst die Ergebnisse der Untersuchungen der Sphere-Funktion und von Ackleys Funktion näher betrachtet. Diese beiden Testfunktionen wurden ausgewählt, da sie unterschiedliche Funktionenklassen repräsentieren. Die unimodale Sphere-Funktion stellt einen einfachen Testfall für den Optimierungsalgorithmus dar, während Ackleys Funktion einen multimodalen Funktionsverlauf aufweist und folglich wesentlich schwieriger zu optimieren ist. Die Betrachtung der zweitgenannte Funktion dient der Überprüfung, ob der Algorithmus dazu in der Lage ist, eine global optimale Lösung zu finden und die Suche nicht mit einer „nur" lokal optimalen Lösung vorzeitig abbricht.

Abbildung 4.8 zeigt die Testergebnisse für die 10-dimensionale Sphere-Funktion und die folgende Konfiguration:

$$\begin{gathered} \text{RECOMBINATION} = \text{ESDiscreteRecombination} \\ (\text{Basiskonfig.}, \text{ADAPTATION}, \text{PNUMBEROFPARENTS}, \text{HOLDFIRSTPARENT}) \\ = (\text{GA}, 0, 1 + \text{DIMENSION}, 1) \end{gathered}$$

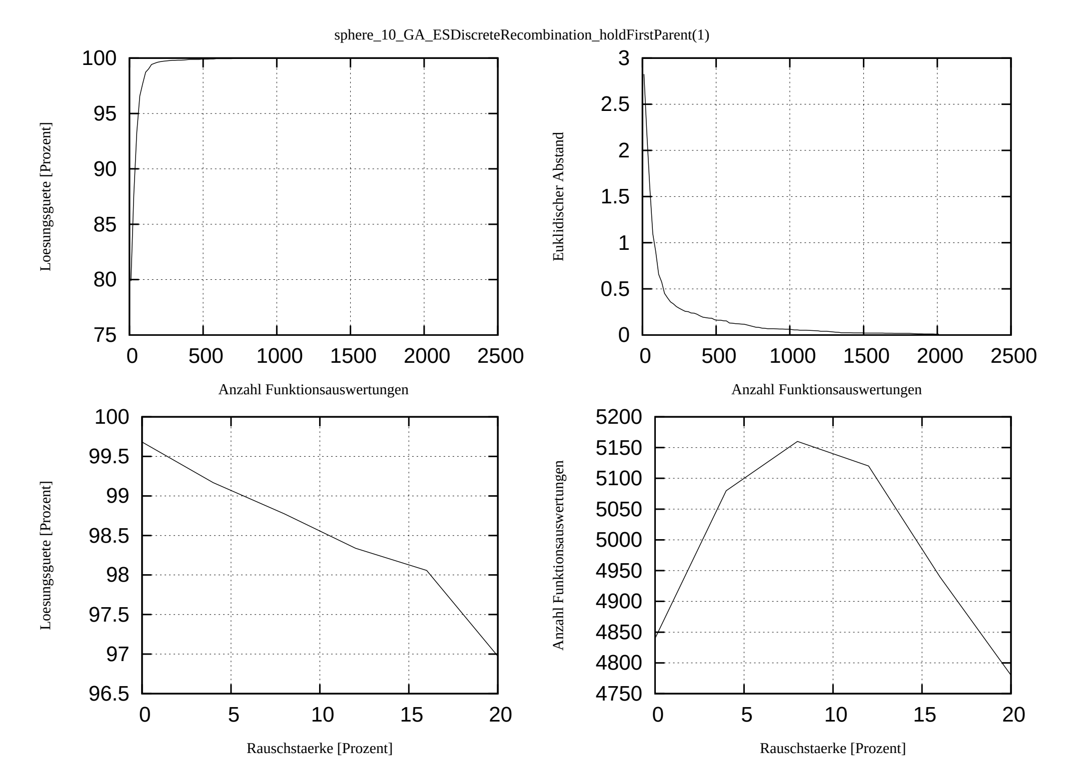


Abb. 4.8.: Testergebnisse für die 10-dimensionale Sphere-Funktion

Die folgenden Ausführungen beziehen sich hauptsächlich auf Abbildung 4.8 l.u. Die genannte Konfiguration stellte sich als die beste der untersuchten heraus. Die anderen unter den Punkten (4) – (6) genannten Einstellungen erwiesen sich bei Verwendung der GA-Grundkonfiguration ebenfalls als hilfreich. Ihre Verwendung führte nur zu unwesentlich schlechteren Ergebnissen. Die unter den Punkten (2) und (3) erwähnten Konfigurationen lieferten jedoch wesentlich schlechtere Ergebnisse. Die Güte der gefundenen Lösung lag in diesen Fällen meist unter 90 Prozent. Das Ergebnis erwies sich bei Verwendung der unter Punkt (3) genannten Einstellungen als etwas besser als bei Verwendung der unter (2) genannten. Ob die Variation der Freiheitsgrade der entsprechenden Operatoren eine wesentliche Verbesserung der Performanz bewirkt, wurde nicht im Detail untersucht. In [2] wird beispielsweise berichtet, daß eine Erhöhung der Anzahl der Schnittstellen zu einem verbesserten Verhalten führen kann. Das erscheint natürlich, da die optimale Anzahl von Schnittstellen sowohl von der Anzahl der Bits, mit denen jede Koordinate repräsentiert wird, als auch der Dimension des Suchraums abhängen sollte und hier mit 1 bzw. 2 möglichweise zu klein ist. Des Weiteren zeigen die Testergebnisse, daß die Durchmischung der Bits vor dem eigentlich Rekombinationsvorgang und die anschließende Rückgängigmachung sich eher nachteilig auswirken. Bemerkenswert ist die Tatsache, daß die Lösungsgüte ohne Verwendung eines Rekombinationsoperators zwischen 94.5 und 98.4 Prozent lag. Das bedeutet, daß die Verwendung der eben erwähnten Operatoren die Suche nach der optimalen Lösung verlangsamt und in dieser Form nicht empfehlenswert

erscheint.

Abbildung 4.8 l.o. und r.o. verdeutlicht, daß die genannte Konfiguration auf lange Sicht dazu in der Lage ist, das globale Optimum der Sphere-Funktion genau zu lokalisieren.

Die Verwendung der ES-Grundkonfiguration führte zu ähnlichen Ergebnissen. Die unter den Punkten (4) – (7) genannten Operatoren erwiesen sich als gut und die unter (2) und (3) genannten als schlecht. Die allgemein als hilfreich geltende Verwendung des unter Punkt (7) genannten Operators führte nicht zu einer Verbesserung des Ergebnisses. Die paarweise diskrete Rekombination der Koordinaten und die globale intermediäre Rekombination der Strategieparameter scheinen also nicht in besonderem Maße empfehlenswert. Das kann damit zusammenhängen, daß keine ausreichende Anzahl von Generationen berechnet wurde und sich somit die Selbstadaption der Strategieparameter (noch) nicht (positiv) auswirkte. In den Optimierungsläufen wurden durchschnittlich nur zwischen 12.45 und 16.80 Durchläufe des Evolutionszyklus durchgeführt (je nach Stärke des Rauschens). Die Verwendung keines Rekombinationsoperators führte hier jedoch zu einer erheblichen Verschlechterung der Ergebnisgüte, die zwischen 81.3 und 84.1 Prozent lag. Diese Beobachtung kann mit der standardmäßigen Verwendung der uniformen Elternselektion erklärt werden, da so bei der Selektion der Nachkommen die Fitneß der einzelnen Individuen unberücksichtigt blieb. Dieses Vorgehen ist bei der Optimierung einer unimodalen Funktion natürlich unvorteilhaft.

Die Verringerung der Dimension auf 2 änderte nichts an den oben beschriebenen Beobachtungen. Sie führte jedoch dazu, daß sich die Ergebnisse der unterschiedlichen Testläufe weniger unterschieden.

Zusammenfassend kann gesagt werden, daß eine Erhöhung der Stärke des gaußschen Rauschens zu einer Verschlechterung der Lösungsgüte führt, die Anzahl der durchgeführten Funktionsauswertungen jedoch von der Stärke des Rauschens unabhängig zu sein scheint. Beide Aussagen können damit erklärt werden, daß die Unsicherheit des Schätzwerts der Lösungsgüte einer bestimmten Lösung keine Berücksichtigung findet. Des Weiteren kann man feststellen, daß die Verwendung des GA-Rahmenalgorithmus zu besseren Ergebnissen als die des ES-Rahmenalgorithmus führte. Diese Tatsache liegt in der verwendeten Art der Elternselektion oder dem verwendeten Mutationsoperator begründet.

Abbildung 4.9 zeigt die Testergebnisse für Ackleys 10-dimensionale Funktion und die folgende Konfiguration:

$$\begin{gathered}\text{RECOMBINATION} = \text{ESDiscreteRecombination}\\ (\text{Basiskonfig.}, \text{ADAPTATION}, \text{PNUMBEROFPARENTS}, \text{HOLDFIRSTPARENT})\\ = (\text{GA}, 0, 1 + \text{DIMENSION}, 1)\end{gathered}$$

Auch hier hat sich die bereits oben erwähnte Konfiguration als die beste erwiesen. Die Ergebnisse verhalten sich hier ähnlich wie die für die Sphere-Funktion beschriebenen. Insgesamt sind die Ergebnisse jedoch schlechter. Das läßt sich mit dem komplizierteren Funktionsverlauf erklären.

Die Betrachtung der anderen sieben Testfunktionen bestätigt die für die Sphere-Funktion und Ackleys Funktion beschriebenen Resultate.

Zusammenfassend kann gesagt werden, daß sich die globale diskrete Rekombination als hilfreichste Form der Rekombination erwiesen hat. Andere Formen der diskreten und

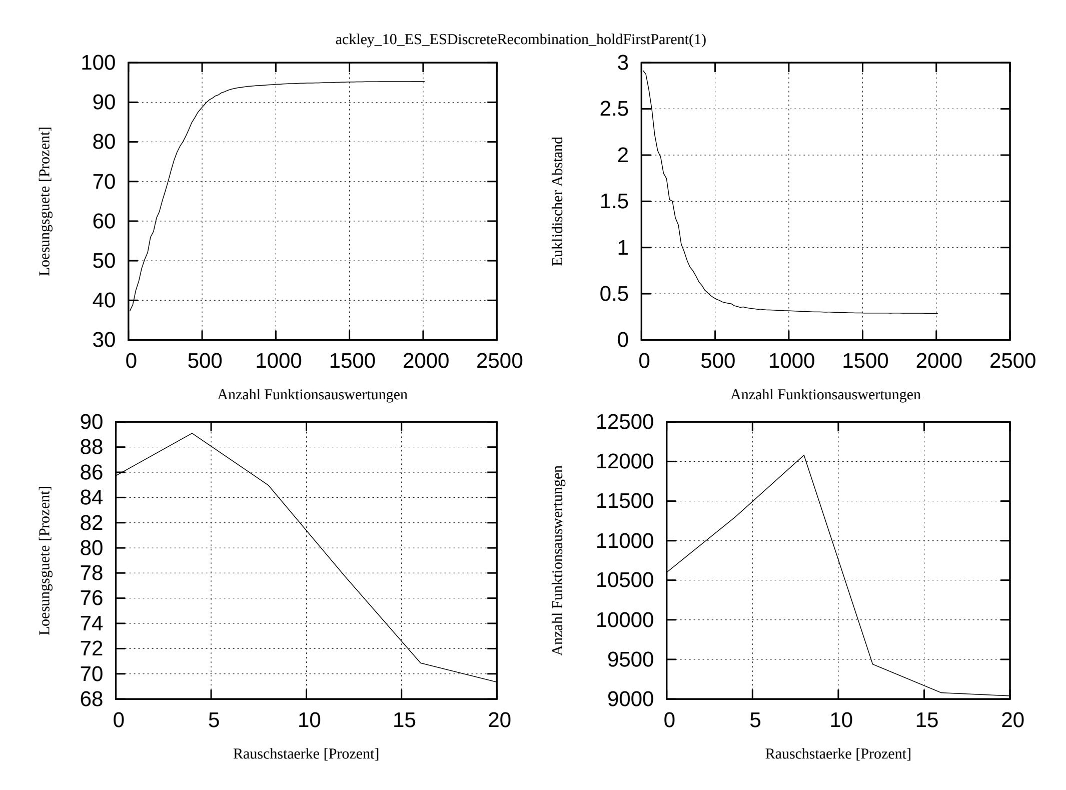


Abb. 4.9.: Testergebnisse für Ackleys 10-dimensionale Funktion

der intermediären Rekombination sind ebenfalls sinnvoll. Die unter den Punkten (2) und (3) beschriebenen Rekombinationsoperatoren erwiesen sich als nicht hilfreich. Des Weiteren wurde festgestellt, daß die positive Wirkung der selbstadaptiven Anpassung der Strategieparameter gering zu sein scheint. Außerdem scheint die Bedeutung des Rekombinationsoperators bei Verwendung der GA-Grundkonfiguration einen relativ kleinen Stellenwert zu besitzen. Ein kompletter Verzicht auf das Konzept der Rekombination hat die Ergebnisse hier nur minimal verschlechtert. Für die klassische Evolutionsstrategie ist das Konzept der Rekombination hingegen zentral.

Nicht diskutiert wurden in diesem Abschnitt beispielsweise die Auswirkungen der Variation der Rekombinationswahrscheinlichkeit oder der Behandlung ungültiger (außerhalb des Definitionsbereichs liegender) Individuen.

4.4.5. Mutationsoperatoren

Peter Kissmann

Bei der Untersuchung der Mutationsoperatoren wurden die drei implementierten Verfahren mit unterschiedlichen Parametereinstellungen untersucht:

k-**Bit-Mutation:** Hier wurden die Mutationswahrscheinlichkeit sowie die Anzahl zu kippender Bits variiert. Für die Mutationswahrscheinlichkeit wurden die Werte 0.0,

was einem Algorithmus ohne jegliche Mutation entspricht, 0.5, 0.75 sowie 1.0 eingestellt. Bei der Anzahl zu kippender Bits wurden die Werte 1, 2, 5 und 10 untersucht.

Standardbit-Mutation: Bei diesem Operator wurde einzig die Wahrscheinlichkeit, ein Bit zu kippen, geändert. Der Standardwert hierfür ist $\frac{1}{n \cdot b}$, wobei n der Dimension des Suchraumes und b der Anzahl Bits, die eine Koordinate kodieren, entsprechen. Als Werte für die Untersuchung wurden der Standardwert, sowie das vier- und sechzehnfache davon eingestellt.

Mutation durch Addition eines Mutationsvektors: Da dies der einzige Mutationsoperator ist, der auf der reellwertigen Darstellung operiert und somit (wegen der Art der Kodierung) der einzige ist, der Individuen außerhalb des zulässigen Bereiches erzeugen kann, wurde hier der Parameter, der angibt, wie mit ungültigen Individuen zu verfahren ist, verändert. Ebenfalls wurde der Parameter, der die Art der Adaption angibt, angepaßt. Für den erstgenannten Parameter wurden die beiden zulässigen Einstellungen durchprobiert: Beibehalten ungültiger Individuen sowie Zerstören und Neuerstellen ungültiger Individuen. Als Adaption wurden die Selbstadaption der Mutationsstärken, mit und ohne Adaption der Rotationswinkel, sowie die kumulative Adaption, ebenfalls mit und ohne Adaption der Rotationswinkel, untersucht.

Bei der Untersuchung der einzelnen Parametereinstellungen wurde dann, wie in der Einleitung dieses Kapitels beschrieben, vorgegangen.

4.4.5.1. Schwierigkeiten bei der Auswertung

Bei der Untersuchung sowohl der verrauschten als auch der nicht-verrauschten Zielfunktionen haben sich einige Probleme herausgestellt, die die Auswertung erheblich erschwerten:

- In nahezu allen Fällen waren die Ergebnisse, die auf der GA-Konfigurationsdatei beruhten, deutlich schlechter als jene, welche auf der ES-Konfigurationsdatei beruhten. So kamen, insbesondere in Gegenwart von Rauschen, die Ergebnisse des GA bestenfalls auf 70 bis 80 Prozent an das Optimum heran, teilweise sogar nur auf 50 bis 60 Prozent. Die Ergebnisse der ES hingegen kamen oftmals auf deutlich über 90 Prozent, teilweise sogar bis auf über 99 Prozent an das Optimum heran.

- Parametereinstellungen, die für eine Funktion bei einer Dimension gut waren, waren teils bei anderen Dimensionen deutlich schlechter als die anderen. So ließ sich leider auch kein allgemeiner Schluß für eine Funktion ziehen, der besagen würde, welche Einstellung hier stets zu guten Ergebnissen führt.

- Bei verrauschten Funktionen kam noch hinzu, daß Einstellungen, die zwar nah ans Optimum herankamen, sehr häufig ausgewertet werden mußten; teilweise bis zu dreimal so oft wie welche, die einen deutlich schlechteren Wert lieferten. Entsprechend führten Einstellungen, die nur relativ selten ausgewertet wurden, bis der

Algorithmus abbrach, zumeist nicht sehr nahe an das Optimum heran; bestenfalls auf knapp 90 Prozent, in der Regel jedoch eher auf etwa 70 bis 80 Prozent. Man muß also unterscheiden, ob man möglichst wenige Auswertungen bevorzugt, oder ob das Ergebnis möglichst dicht an das Optimum heranreichen soll.

4.4.5.2. Ergebnisse der Untersuchungen

Trotz dieser Schwierigkeiten wurde versucht, die Ergebnisse auszuwerten. Bei den meisten Funktionen waren sie jedoch dermaßen uneindeutig, daß hier nicht weiter darauf eingegangen wird, sondern nur ein paar scheinbar klarere Ergebnisse aufgezeigt werden.

Für die meisten nicht-verrauschten Funktionen hat sich gezeigt, daß für die Mutationsoperatoren, die auf der Binärdarstellung der Koordinaten operieren (also k-Bit- und Standardbit-Mutation), der Algorithmus eine Lösung lieferte, die deutlich näher (sowohl, was den Funktionswert als auch den Euklidischen Abstand betrifft) an das Optimum heranreichte als für den Mutationsoperator, der auf der reellwertigen Darstellung arbeitet (siehe Abbildung 4.10).

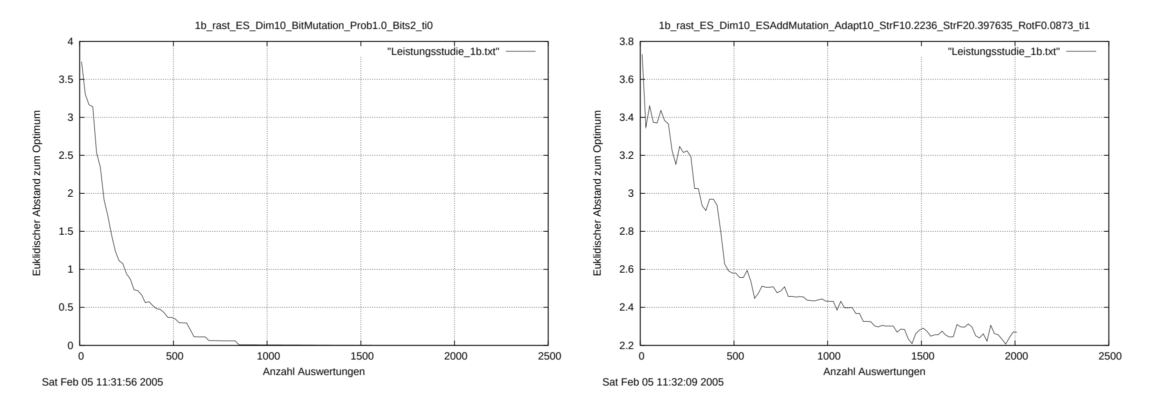


Abb. 4.10.: Häufiges Verhalten bei nicht-verrauschten nicht-trivialen Funktionen (hier am Beispiel von Rastrigins Funktion): Mutationsoperatoren auf Binärdarstellung (links) deutlich näher am Optimum als Mutationsoperatoren auf reellwertiger Darstellung (rechts).

Bei Schaffers Funktion war zu beobachten, daß, im Falle der ES-Konfigurationsdatei die 1- und 2-Bit-Mutationen deutlich besser waren als alle anderen Verfahren. Sobald aber die GA-Datei als Grundlage verwendet wurde, kehrte sich dieses Bild zumindest teilweise wieder um: Die 1-Bit-Mutation war in diesem Fall schlechter als die meisten anderen Einstellungen. Dies mag daran liegen, daß beim Zusammenspiel von 1-Punkt-Crossover und 1-Bit-Mutation nur sehr geringe Änderungen an den Koordinaten der Individuen vorgenommen werden. Dadurch ist es ihnen nicht mehr möglich, ein lokales Optimum zu verlassen und der Algorithmus fährt sich dort fest. Wird hingegen der Schwefel Rekombinationsoperator verwendet, werden die Koordinaten anscheinend soweit verändert, daß die geringe Änderung, die die 1- oder 2-Bit-Mutation mit sich bringt, ausreicht, um sehr schnell zum Optimum zu gelangen. Dieses Bild hat sich sowohl bei

nicht-verrauschten (siehe Abbildung 4.11) als auch bei verrauschten Funktionen gezeigt.

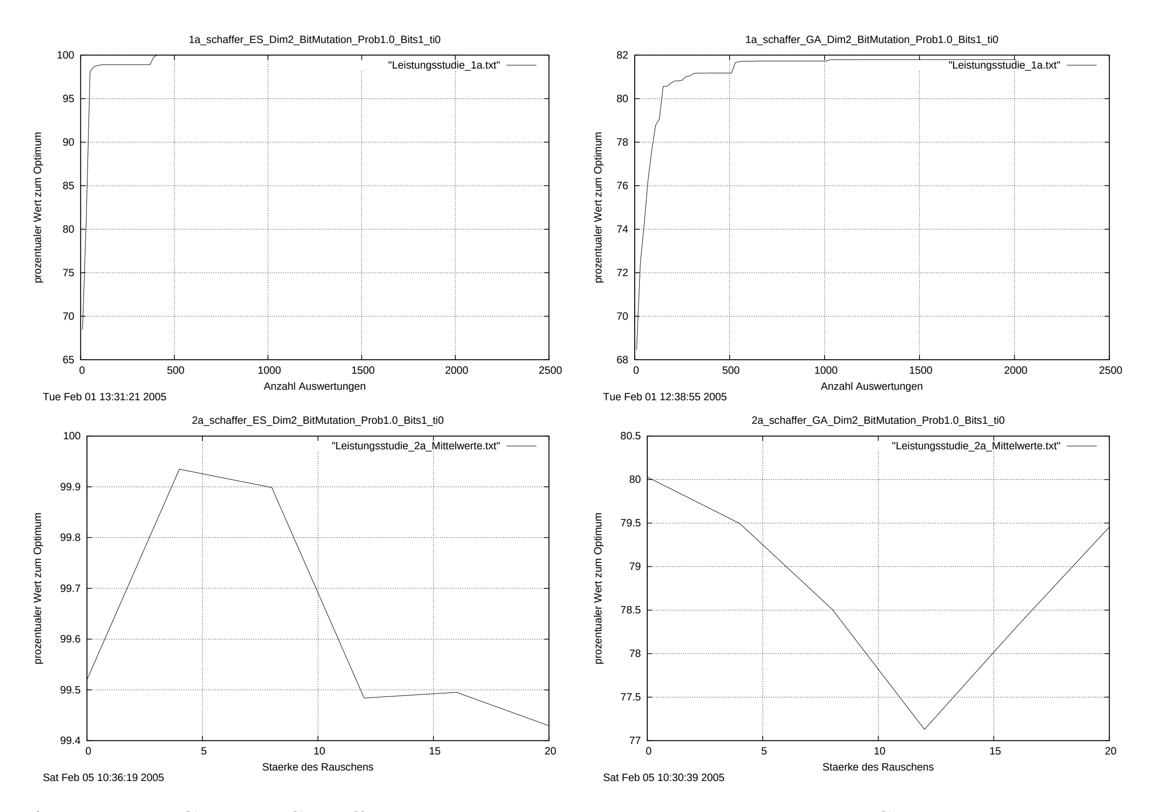


Abb. 4.11.: Gut bei Schaffers Funktion: 1-Bit-Mutation mit der ES-Datei sowohl ohne (oben links) als auch mit Rauschen (unten links). Schlecht hingegen: 1-Bit-Mutation mit der GA-Datei sowohl ohne (oben rechts) als auch mit Rauschen (unten rechts).

Für die verrauschte Sphäre hat sich für die Operatoren, die auf den binär-kodierten Koordinaten operieren, gezeigt, daß sowohl in hohen wie auch in niedrigen Dimensionen eine größere Änderung der Koordinaten, etwa bei der 10-Bit-Mutation oder der Standardbit-Mutation mit dem 16-fachen der standardmäßigen Kipp-Wahrscheinlichkeit, im Falle der ES-Datei als Grundlage zu besseren Ergebnissen führte. Demgegenüber führte eine geringere Änderung, etwa bei der 1-Bit-Mutation oder der Standardbit-Mutation mit standardmäßiger Kipp-Wahrscheinlichkeit, im Falle der GA-Datei als Grundlage zu besseren Ergebnissen. Aber auch der Operator, der auf der reellwertigen Koordinatendarstellung arbeitet, hat sich durchaus bewährt: Bei geringen Dimensionen in Verbindung mit der Grundlage der GA-Datei hat sich gezeigt, daß die kumulative Adaption, egal ob mit oder ohne Adaption der Rotationswinkel, besser war als alle anderen Verfahren. Mit der ES-Datei war diese Einstellung allerdings nicht ganz so gut. In höheren Dimensionen hat sich dieses Bild weitestgehend bestätigt: Die kumulative Adaption führte wieder mit am Schnellsten zum Erfolg, sofern die GA-Konfigurationsdatei zugrunde gelegt wurde. Aber auch die Selbstadationsverfahren, ebenfalls mit und ohne Adaption

der Rotationswinkel, haben sich hier bewährt, jedoch nur, wenn ungültige Individuen verworfen und neu erzeugt wurden. Anderenfalls hat sich die Population vermutlich zu lange stets in die verkehrte Richtung entwickelt, was durch das Verwerfen ungültiger Individuen weitestgehend unterbunden wurde. Im Falle der ES-Datei hat sich nun auch die Selbstadaption der Mutationsstärken zusammen mit der Adaption der Rotationswinkel als recht vielversprechend erwiesen. In diesem Fall führten allerdings sowohl die Erzeugung neuer Individuen als auch die Beibehaltung der alten, wenn diese ungültig waren, gleichermaßen zum Erfolg. In allen Fällen wurde zwar das Optimum relativ gut approximiert (nicht eine Einstellung führte nach Ende des Algorithmus zu einem Wert, der weniger als 90 Prozent des Optimums entspricht), aber bei den gerade beschriebenen Einstellungen war die Anzahl der Funktionsauswertungen deutlich geringer als bei den restlichen (einige Beispiele siehe Abbildung 4.12).

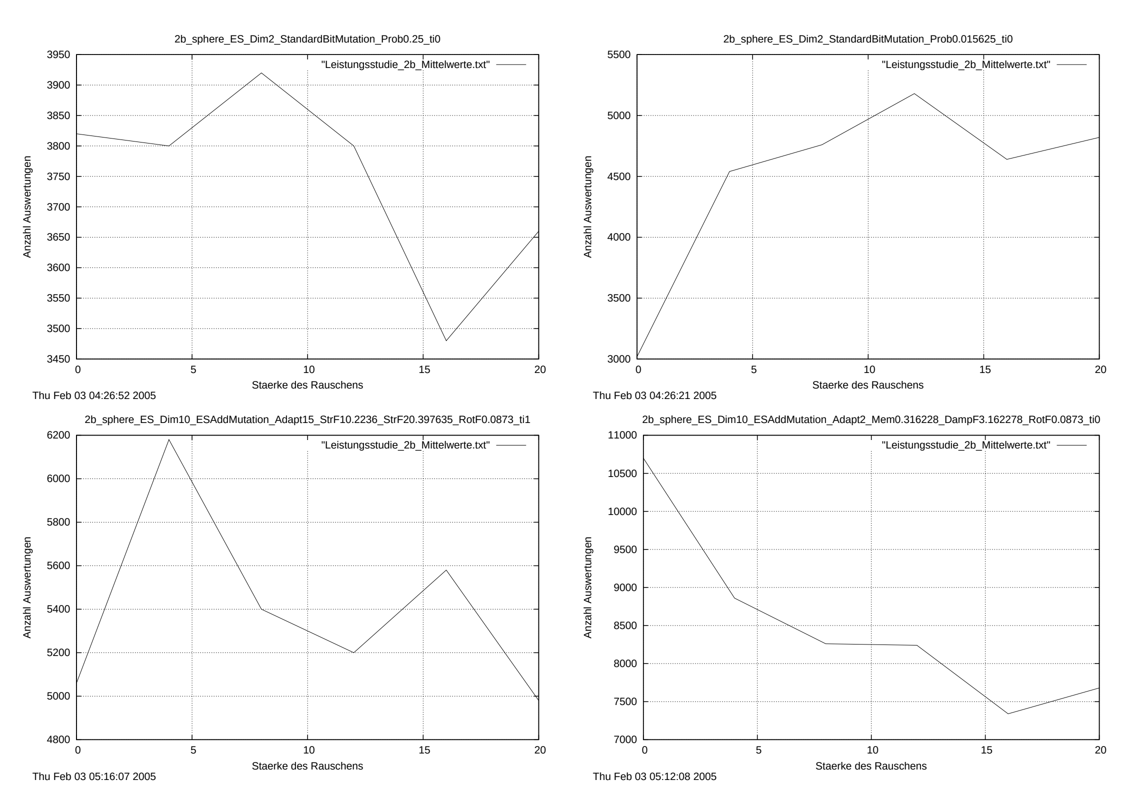


Abb. 4.12.: Beispiele zur verrauschten Sphäre mit der Grundlage der ES-Konfigurationsdatei, oben Dimension 2 und unten Dimension 10. Oben links Standardbit-Mutation mit 16-fachem der Standard-Kipp-Wahrscheinlichkeit (gut), oben rechts mit der Standard-Kipp-Wahrscheinlichkeit (schlecht). Unten links Mutation durch Addition eines Mutationsvektors mit Selbstadaption der Mutationsstärken und Adaption der Rotationswinkel sowie Verwerfen ungültiger Individuen (gut), unten rechts mit kumulativer Mutationsstärkenadaption und Beibehalten ungültiger Individuen (schlecht)

4.4.5.3. Fazit

Es hat sich insgesamt also ein recht widersprüchliches Verhalten gezeigt: Die besten Operatoren hängen nicht nur von der auszuwertenden Funktion ab, sondern auch von der Dimension des Suchraumes.

Auch die verwendete Konfigurationsdatei hatte einen ungeheuer hohen Einfluß auf die Ergebnisse. Letzteres lag vermutlich daran, daß der verwendete 1-Punkt-Crossover erheblich schlechter abgeschnitten hat als viele der anderen Rekombinationsoperatoren, wie im entsprechenden Abschnitt erwähnt wurde. Aufgrund dieses Verhaltens ist es nicht möglich, einen optimalen Operator anzugeben, der für alle Fälle gute Ergebnisse liefert.

Die Leistungsstudien haben auch nicht geholfen, zu entscheiden, welche Parametereinstellungen bei den einzelnen Operatoren zu guten Ergebnissen führen. Auch hier gab es deutliche Unterschiede, je nachdem, welche Dimension welcher Zielfunktion gerade untersucht wurde.

Eine Sache, die aus den Diagrammen nicht hervorgeht, ist ein Problem, das nur während des Beobachtens des Algorithmusdurchlaufes auffiel: Das Verwerfen ungültiger Individuen, das hier nur bei dem Operator, der auf der reellwertigen Kodierung arbeitet, angewendet wurde, hat in einigen Fällen zu einer ungeheuer langen Laufzeit geführt. Insbesondere, wenn kumulative Adaption in Verbindung mit der Adaption der Rotationswinkel durchgeführt wurde, trat dieses Problem einige Male deutlich zu Tage. Aus irgendeinem Grund stieg die Mutationsstärke recht stark an (vermutlich haben sich die Individuen zunächst alle in etwa in die gleiche Richtung bewegt), so daß sie bereits nach drei Generationen so hoch war, daß die Wahrscheinlichkeit, ein gültiges Individuum zu erzeugen, verschwindend gering war. In den meisten Fällen aber wurde die Mutationsstärke relativ schnell recht gering, so daß die Individuen großteils nur noch kleine Schritte machten und somit nicht aus dem zulässigen Bereich herausliefen. Da die ungültigen Individuen bereits vor der Auswertung der Funktion verworfen wurden, konnte sich dieses natürlich nicht in den Graphen widerspiegeln.

4.4.6. Terminierung

Igor Gudovsikov

Zu den Zielen der Leistungsstudien gehörte die Untersuchung der Terminierung von Evolutionären Algorithmen bei Plus- und Kommaselektionsstrategien. Die Terminierung wurde dabei mit zwei Methoden untersucht:

- Veränderung des Parameters „Anzahl Schritte ohne Verbesserung“ und
- Veränderung des Epsilon-Parameters (Differenz der Fitneßwerte)

Die Anzahl der Auswertungen für ein Individuum wurde auf 20 eingestellt, so daß bei der Auswertung der ersten Generation die gesamte Anzahl der Auswertungen immer 200 war. Bei allen nachfolgenden Generationen brauchte man jeweils 400 Auswertungen.

Als kleine Einführung zeigen wir am Beispiel der Sphere-Funktion der Dimension 2, wie die Untersuchungen ohne Rauschen durchgeführt wurden.

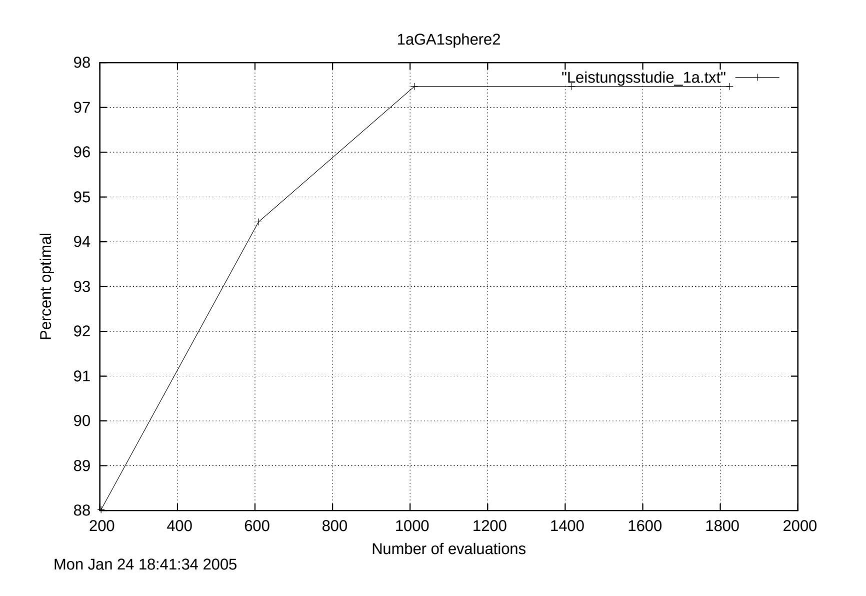


Abb. 4.13.: Sphere: Anzahl Schritte ohne Verbesserung: 2

Die hier präsentierten Abbildungen zeigen den Responsewert der besten bisher ermittelten Lösung als Prozentzahl des bekannten Optimums. Jeder Punkt auf einer Graphik ist der Responsewert einer einzelnen Generation. In Abbildung 4.13 ist die Schrittanzahl ohne Verbesserung auf 2 eingestellt.

Es ist hier deutlich zu sehen, daß sich nach zwei Generationen der Response Wert nicht verbessert hat. Der Algorithmus terminiert an dieser Stelle und das beste Individuum, das wir bekommen haben, hat einen Wert von 97, 5% des Optimums.

Im Folgenden versuchen wir die Anzahl der Schritte zu erhöhen. In Abbildung 4.14 ist dieser Parameter auf 4 eingestellt. Man sieht, daß es einen „Sprung“ in Richtung des Optimums gibt. Aktuell sind wir nur 1% vom optimalen Wert entfernt.

Wie in Abbildung 4.15 zu sehen ist, bringt die Erhöhung des Parameters auf 6 uns nicht weiter.

Erst bei 10 Schritten ohne Verbesserung kommen wir zu 100% an das Optimum heran (vergleich Abbildung 4.16).

Anhand dieser Methode wurde versucht, für jede Benchmarkfunktion der kleinste Wert der Schrittanzahl zu bestimmen, so daß der EA-Algorithmus das Optimum findet. Es wurden dabei einige Zusammenhänge festgestellt:

Schluß 1. *„Bei den Funktionen, die nah am Optimum steil sind, kommen Evolutionäre Algorithmen mit einer kleineren Anzahl von Schritten ohne Verbesserung an das Optimum heran.“*

Dieser Zusammenhang wurde bei mehreren Graphiken beobachtet. Wir zeigen das an einem Beispiel von *Ellipsoid* und *Schwefels Ridge* Funktionen. Die beiden Funktionen haben für uns keine sehr wichtigen Unterschiede. Die einzige Ausnahme ist, daß *Schwefels Ridge* Funktion nah am Minimum steil ist, während Ellipsoid nah an Minimum flach ist.

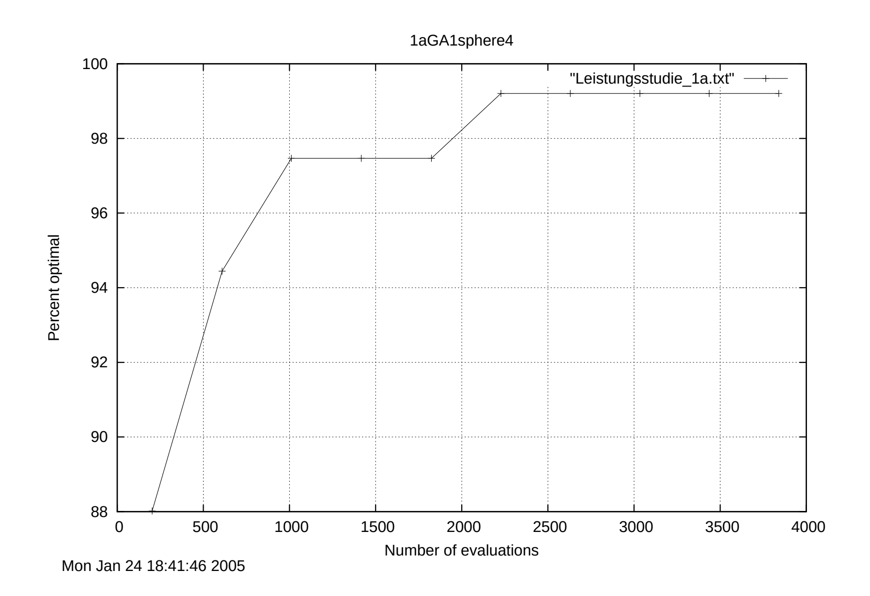


Abb. 4.14.: Sphere. Anzahl Schritte ohne Verbesserung: 4

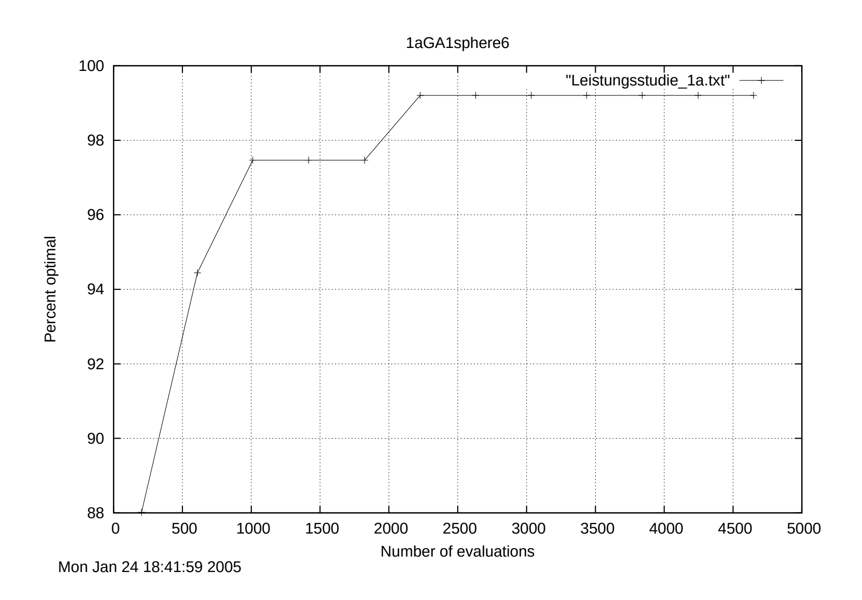


Abb. 4.15.: Sphere. Anzahl Schritte ohne Verbesserung: 6

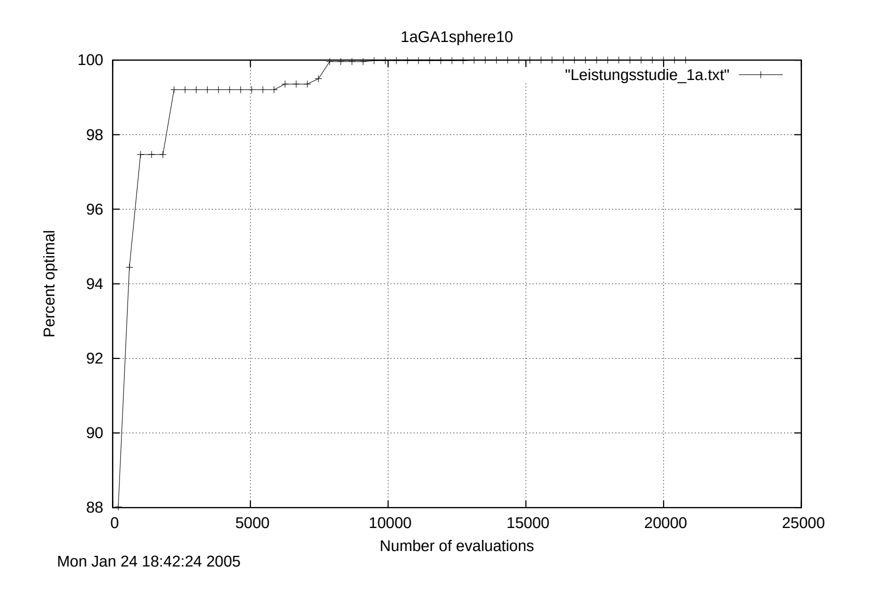
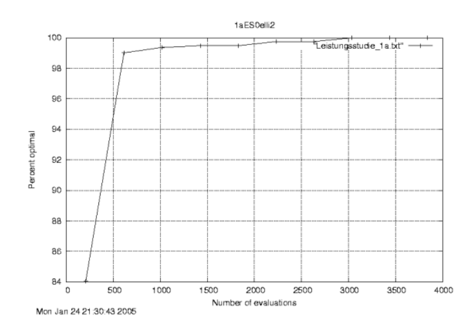


Abb. 4.16.: Sphere. Anzahl Schritte ohne Verbesserung: 10

Bei der *Ellipsoid* Funktion wurde hier die Schrittenanzahl auf 2 eingestellt. Man kann beobachten (vergleiche Abbildung 4.17), daß der Algorithmus schon in der zweiten Generation auf 99% an das Optimums herankommt. Danach „klettert“ der Responsewert langsam zum Optimum. Das heißt, daß eine Terminierung bei 2 Schritten ohne Verbesserung für diese Funktion ausreichend sind, um das Optimum zu erreichen! Ganz anders sieht es im Fall von *Schwefels Ridge* Funktion aus. Im Bereich von ca. 2000 bis 5500 Funktionsauswertungen wird keine Verbesserung beobachtet. Das bedeutet, daß in diesem Fall sogar bei 7 Schritten ohne Verbesserung der Algorithmus nur sehr nah am Optimum terminieren würde. Um das Minimum hier zu erreichen, wurden 8 Schritte ohne Verbesserung benötigt.

In diesem Zusammenhang wurde empirische Werte für mehrere Benchmarkfunktionen (in diesem Fall Dimension 2) bestimmt, mit denen der Algorithmus höchstwahrscheinlich das Minimum findet:

- *Step*: 1 oder 2 Schritte
- *Ellipsoid*: 2 Schritte
- *Rosenbrock*: 2 Schritte
- *Sphere*: 4 Schritte
- *Griewank*: 4 Schritte
- *Ackley*: 6 Schritte
- *Ridge*: 8 Schritte



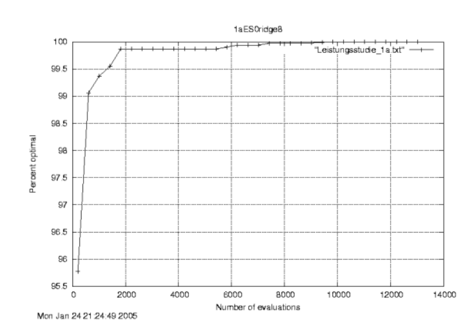
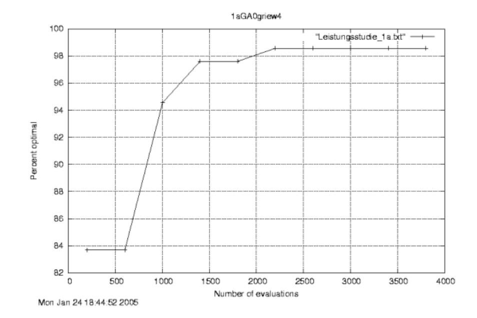


Abb. 4.17.: Terminierung: Ellipsoid mit der Anzahl Schritte 2 und Schwefels Ridge Funktionen mit der Anzahl Schritte 8

Bei der Untersuchung von *Schaffers* und *Rastrigins* Funktionen sind wir noch zu einem anderen interessanten Schluß gekommen.

Schluß 2. *„Bei den Funktionen, die wenig lokale Minima oder Maxima haben, kommen Evolutionäre Algorithmen ebenfalls mit einer kleineren Anzahl von Schritten ohne Verbesserung aus.“*

Eine vernünftige Erklärung dafür könnte sein, daß der Algorithmus in einem lokalen Minimum oder Maximum stecken bleibt, und erst mit der Erhöhung der Schrittanzahl steigt die Wahrscheinlichkeit, daß der Algorithmus aus dem lokalen Optimum herauskommt.



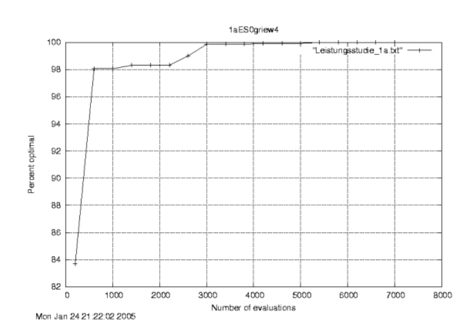


Abb. 4.18.: Terminierung: Vergleich von Genetischen Algorithmen und Evolutionären Strategien

Beim Vergleich von Ergebnissen zwischen Komma-Selektion und Plus-Selektion wurden keine großen Unterschiede festgestellt. Die Theorie besagt, daß die Komma-Selektion

besser zu verwenden ist, wenn eine Funktion mehrere lokale Optima besitzt, aber es wurden keine Ergebnisse hergeleitet, die diese Annahme beweisen würden.

Beim Vergleich von Genetischen Algorithmus und Evolutionsstrategie wurden ebenfalls keine großen Unterschiede festgestellt. Allerdings war die Evolutionsstrategie generell besser. Bei gleicher Anzahl von Schritten ohne Verbesserung hat der Genetische Algorithmus nicht immer das Optimum erreicht. (z. B. Abbildung 4.18)

4.4.7. 20-fache Lösungsauswertung versus Auswahlverfahren nach Koenig und Law

Christian Horoba

In diesem Abschnitt werden zwei Möglichkeiten der Selektion viel versprechender Individuen miteinander verglichen.

Als erste Möglichkeit wurde der einfache Fall der 20-fachen Auswertung jeder Lösung und der anschließenden Schätzung des Erwartungswerts betrachtet. In diesem Fall werden die besten außerhalb der Eltern- und Nachkommenpopulation zu speichernden Individuen aufgrund dieser Erwartungswertschätzungen ausgewählt.

Als zweite Möglichkeit wurde das kompliziertere statistische Auswahlverfahren nach Law und Koenig verwendet (siehe 6.2.6). Hierbei wurde die Anzahl der initialen Auswertungen n_0 auf 20 gesetzt (unabhängig von der Stärke des Rauschens). Der Gleichheitsbereich-Parameter (indifference-zone parameter) δ^* wurde auf das 0.01-fache des maximalen Funktionswerts auf dem betrachteten Definitionsbereich gesetzt. Die Wahrscheinlichkeit für eine korrekte Auswahl von Individuen wurde durch $P^* = 0.75$ nach unten beschränkt. Sei μ der kleinste Erwartungswert der zur Verfügung stehenden Individuen. Eine korrekte Auswahl liegt vor, wenn die Menge der ausgewählten Individuen mindestens ein Individuum enthält, dessen Erwartungswert kleiner als $\mu + \delta^*$ ist.

Die Abbildungen 4.19 und 4.20 zeigen die Testergebnisse für die beiden Selektionsverfahren.

Es fällt auf, daß bei Verwendung einer festen Anzahl von Funktionsauswertungen pro Lösung und zunehmender Stärke des Rauschens, die Güte der gefundenen Lösungen sinkt (99.7967, 99.1528, 98.8471, 98.3963, 97.6121, 97.4683 Prozent), die Anzahl der benötigten Funktionsauswertungen jedoch im Wesentlichen gleich bleibt (4920, 4840, 5440, 5100, 5300, 5080). Bei Verwendung des zweitgenannten Selektionsverfahrens fällt auf, daß die Güte der gefundenen Lösung im Wesentlichen gleich bleibt (99.7576, 99.5012, 99.2479, 99.3656, 99.1546, 99.3465 Prozent), die Anzahl der benötigten Funktionsauswertungen jedoch rapide steigt (4907.55, 10039.4, 37177, 85440.1, 154193, 240822).

Die Anzahl der durchgeführten Funktionsauswertungen erscheint bei stärkerem Rauschen für die Optimierung von Simulationsmodellen unangemessen hoch. Die Einstellung der Parameter n_0, d^* und P^* wurde im Rahmen dieser Studie nicht im Detail untersucht. Möglicherweise kann durch eine geschicktere Wahl das Ergebnis bei Verwendung des zweitgenannten Selektionsverfahrens optimiert werden.

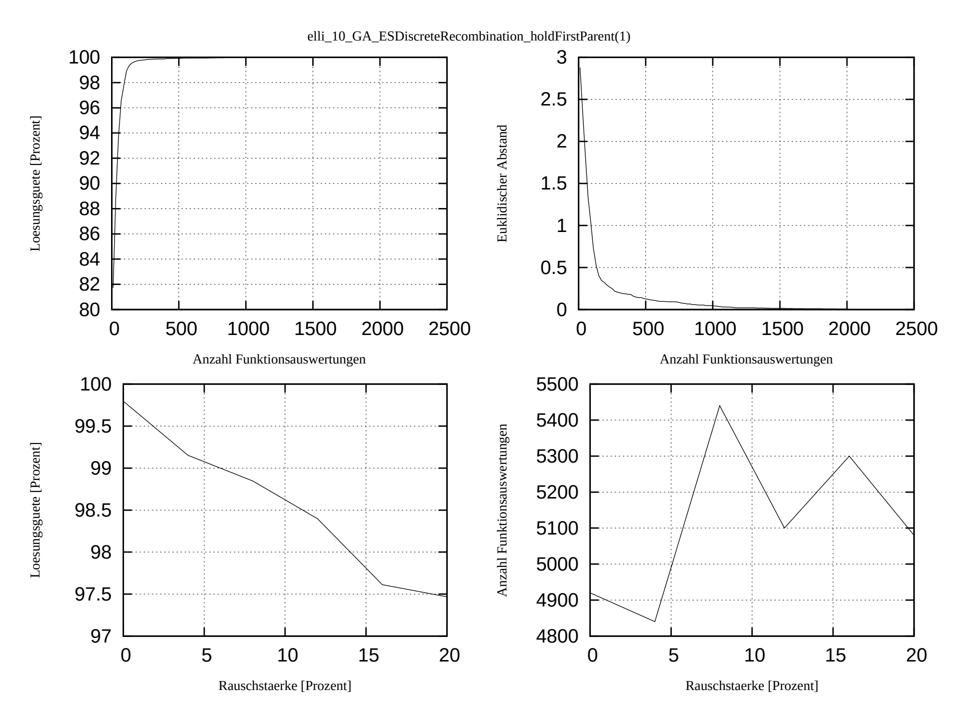


Abb. 4.19.: Testergebnisse für die 10-dimensionale Ellipsoid-Funktion mit fester Anzahl von Auswertungen einzelner Lösungen

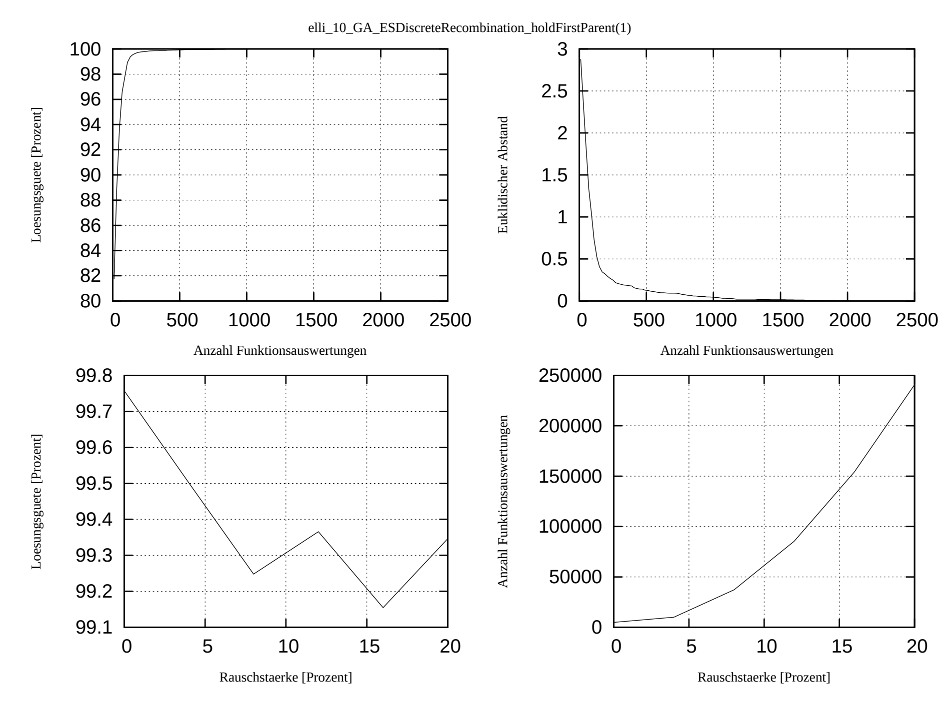


Abb. 4.20.: Testergebnisse für die 10-dimensionale Ellipsoid-Funktion mit variabler Anzahl von Auswertungen einzelner Lösungen

5. Memetische Algorithmen

5.1. Theorie der memetischen Algorithmen

Julia Hielscher

Memetische Algorithmen stellen eine Verbindung zwischen einem evolutionären Algorithmus (siehe Kapitel 4) und einem lokalen Suchverfahren dar. Letzteres ist in unserem Fall die Response Surface Methode (siehe Kapitel 3).
Der Pseudocode für den entstehenden memetischen Algorithmus ist in Abbildung 5.1 dargestellt. Nach jeder der verwendeten Operationen Initialisierung, Mutation und Rekombination wird die lokale Optimierung mit RSM durchgeführt. Durch die Verwendung eines lokalen Suchverfahrens während der Optimerung bestehen die Populationen des evolutionären Algorithmus stets aus lokalen Optima. Stellt sich nach einem Durchlauf des Algorithmus heraus, daß die Population konvergiert ist, daß also viele Individuen nah beieinander liegen und somit vermutlich das gleiche lokale Optimum markieren, so wird erneut mutiert und lokal optimiert um die Diversität wieder zu erhöhen.
Da lokale Suchverfahren per Definition auch nur lokale Optima finden, liegt der Vorteil von memetischen Algorithmen darin, daß sie diesen Nachteil der ansonsten effizienten lokalen Suchstrategien durch die zusätzliche Verwendung von evolutionären Algorithmen ausgleichen. Es entsteht dadurch ein höchst effizientes Optimierungsverfahren.

Ein Spezialfall des memetischen Algorithmus ist die iterierte lokale Suche. Hierbei werden Populationen der Größe 1 verwendet. Das Verfahren arbeitet ohne Rekombination. Zur Optimierung wird also zufällig ein Individuum erzeugt und RSM darauf angewendet. Danach wird das Individuum mutiert, lokal optimiert und mit dem Anfangsindividuum verglichen. Ist bei die Fitneß des neu entstandenen Individuums größer als die Fitneß des Startindividuums, so wird das neue Individuum benutzt um weitere Lösungen durch Mutation und lokale Suche zu generieren. Dieser Vorgang wird wiederholt bis zur Erfüllung der Terminierungsbedingung.
Für den Mutationsoperator wird dabei nicht eine feste Stärke für alle Mutationen verwendet, sondern eine dynamische Anpassung der Mutationsstärken durchgeführt. Dies bedeutet, daß das Verfahren mit kleinen Mutationsstärken beginnt und durch Verdoppeln zu größeren Stärken gelangt, bis bei einer Mutationsstärke von $\frac{1}{2}$ wieder zu den kleinen Stärken zurück gesprungen wird. Die dynamische Anpassung soll das Fehlen der Rekombination ausgleichen und dafür sorgen, daß die Mutation sowohl zur Feinabstimmung als auch zur Erforschung neuer Gebiete im Suchraum dienen kann.

Eine Idee zur weiteren Verbesserung des memetischen Algorithmus besteht in der Einbeziehung des Wissens über die Fitneßlandschaft in den Optimierungsprozeß. Dazu

```
begin
    for j := 1 to Populationsgrösse do
        i := Initialisierung();
        i := RSM(i);
    endfor;
    repeat
        for i := 1 to #Rekombinationen do
            selektiere zwei Elternteile i_a, i_b ∈ P zufällig;
            i_c := Rekombination(i_a, i_b);
            i_c := RSM(i_c);
            füge Individuum i_c zu P hinzu;
        endfor;
        for i := 1 to #Mutationen do
            selektiere ein Idividuum i_d ∈ P zufällig;
            i_m := Mutation(i_d);
            i_m := RSM(i_m);
            füge Individuum i_m zu P hinzu;
        endfor;
        P := Selektion(P);
        if P konvergiert then P := Mutation&RSM(P);
    until Terminierung = wahr;
end;
```

Abb. 5.1.: Pseudocode des Memetischen Algorithmus

wurde das adaptive Verfahren entwickelt. Es beruht auf der Berechnung des Fitneß-Distanz-Korrelationskoeffizienten (FDC) zur Feststellung des Schwierigkeitsgrades einer Fitneßfunktion. Sei eine Menge von Punkten $x_1, x_2, \ldots, x_m$ gegeben, so kann der FDC abgeschätzt werden zu

$$\varrho(f,d) \approx \frac{1}{\sigma(f)\sigma(d)} \frac{1}{m} \sum_{i=1}^{m} (f_i - \overline{f})(d_i - \overline{d})$$

Hierbei gilt:

- $f = f_1, \ldots, f_n$ sind die Fitneßwerte und $d = d_1, \ldots, d_n$ die Distanzwerte,
- σ ist die Standardabweichung,
- m ist die Anzahl der vorgegebenen Punkte,

- $f_i = f(x_i)$ ist der Fitneßwert des Punktes x_i,
- $\overline{f}$ ist der Mittelwert über alle Fitneßwerte,
- $d_i = d_{opt}(x_i)$ ist die kürzeste Distanz von Punkt x_i zum globalen Optimum und
- $\overline{d}$ ist der Mittelwert über alle Distanzwerte.

Der Wert des FDC liegt im Intervall von -1 bis 1. Dabei bedeutet bei einem Minimierungsproblem $\varrho = 1$, daß Fitneß und Distanz zum Optimum eine perfekte Relation bilden und die Suche somit „leicht“ werden könnte. Ein Wert $\varrho = -1$ deutet darauf hin, daß mit steigender Fitneß auch die Distanz zum Optimum steigt und daß es sich somit um ein schwieriges Problem handelt.
Zur Berechnung des FDC ist es normalerweise notwendig, daß das globale Optimum bekannt ist. Da dies in unserem Fall nicht möglich ist, wurde versucht, von einem Teil des Suchraumes auf den gesamten Suchraum zu schließen. Der memetische Algorithmus wird also zunächst eine bestimmte Anzahl von Generationen normal durchgeführt. Danach wird auf Grundlage aller bisher besuchten Punkte der Fitneß-Distanz-Korrelationskoeffizient berechnet. Das beste bisher gefundene Individuum wird dabei als globales Optimum angenommen.
Auf Basis des FDC können dann Anpassungen der Operationen vorgenommen werden, die bei einem bestimmten Schwierigkeitsgrad der Funktion als sinnvoll erscheinen.

5.2. Programmbeschreibung

Katharina Balzer

5.2.1. Anforderungen an den memetischen Algorithmus

In diesem Abschnitt soll kurz beschrieben werden, welche Anforderungen an den memetischen Algorithmus gestellt wurden.

Die Aufgabe war es, einen hybriden Optimierer zu entwickeln, der die beiden Optimierverfahren aus dem ersten Semester, nämlich den evolutionären Algorithmus und die Response Surface Methode, miteinander verbindet. Als beste Lösung für ein hybrides Verfahren stellte sich der memetische Algorithmus heraus. Dieser Algorithmus sollte dann, wie alle anderen Optimierverfahren auch, an eine zentrale Verwaltungsklasse, die *Whitebox*, angebunden werden.
Desweiteren sollten noch einige Spezialfälle des memetischen Algorithmus betrachtet werden. Den wichtigsten stellt das adaptive Verfahren dar, das mit Hilfe des Fitneß-Distanz-Korrelationskoeffizienten den Schwierigkeitsgrad der betrachteten Funktion ermittelt, so daß zur Laufzeit die Parameter so verändert werden, daß das Verfahren effizienter wird. Weitere Spezialfälle sind zum einen die iterierte lokale Suche, bei der ohne Rekombination und mit Populationsgrößen von 1 optimiert wird, zum anderen ein evolutionärer Algorithmus, bei dem nur nach der Initialisierung lokal optimiert wird, der aber sonst wie ein ganz normaler evolutionärer Algorithmus funktioniert.

5.2.2. Aufbau des memetischen Algorithmus

In diesem Abschnitt soll kurz auf den Aufbau des memetischen Algorithmus eingegangen werden. Einen Überblick gibt das Klassendiagramm in Abbildung 5.2.

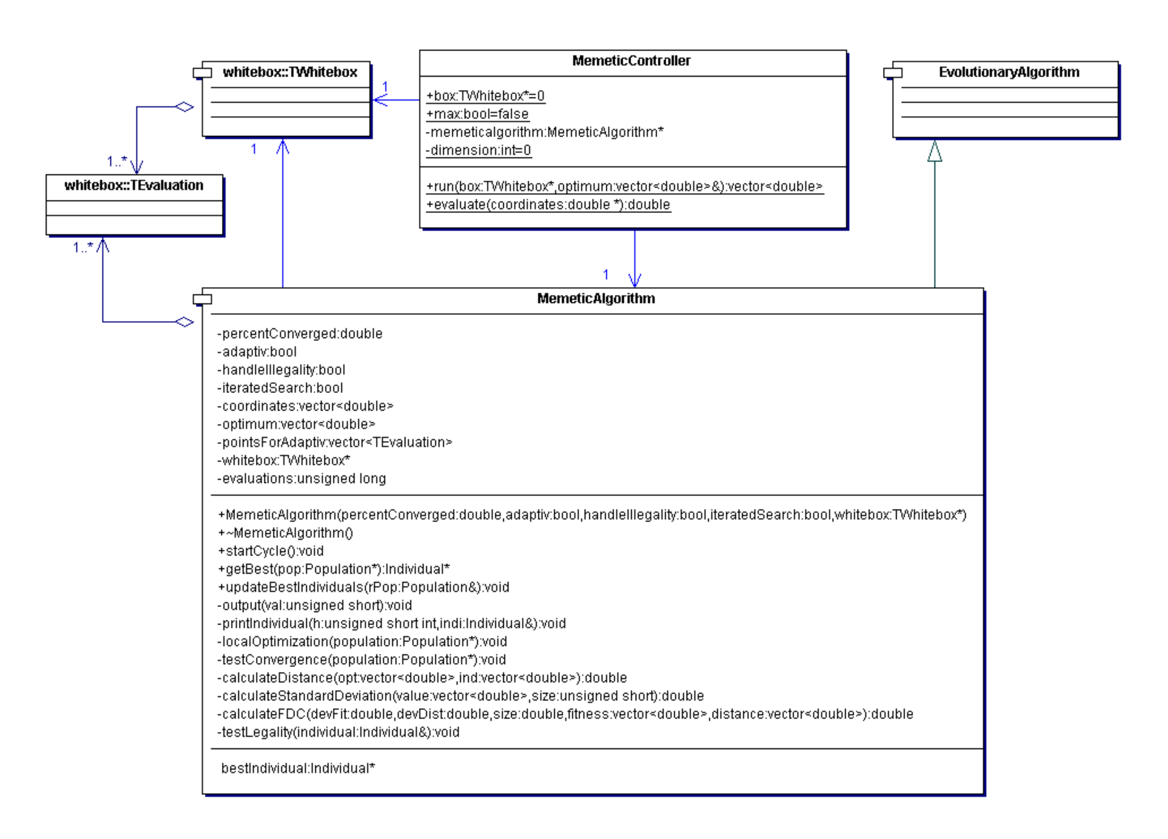


Abb. 5.2.: Klassendiagramm des memetischen Algorithmus

Da der memetische Algorithmus zum größten Teil auch ein evolutionärer Algorithmus ist, erbt er den meisten Teil der Funktionalität vom evolutionären Algorithmus. Allerdings mußte die Methode *startCycle* überladen werden, da im memetischen Algorithmus ja zusätzlich immer die lokale Suche aufgerufen werden muß. Dies geschieht über einen Methodenaufruf in der *Whitebox*. Ansonsten enthält die Klasse *MemeticAlgorithm* noch einige private Methoden, die für den Ablauf des Algorithmus wichtig sind. Eine genauere Erläuterung befindet sich in der Klassenbeschreibung in Abschnitt 5.2.3. Der evolutionäre Algorithmus, von dem der memetische Algorithmus erbt, ist genauso aufgebaut, wie der normale evolutionäre Algorithmus. Der Aufbau des evolutionären Algorithmus kann in Kapitel 4.2.2 nachgesehen werden.

Der Kontrollfluß beim memetischen Algorithmus sieht nun folgendermaßen aus: Die Klasse *MemeticController* holt sich aus der Klasse *Whitebox* die Parameter, die der Benutzer über die GUI eingestellt hat und erzeugt alle Objekte, die der memetische Algorithmus benötigt. Danach wird mit Hilfe der Methode *run* der Kontrollfluß an die

Klasse *MemeticAlgorithm* weitergegeben. Dort wird dann der eigentliche memetische Algorithmus gestartet. Wenn dieser komplett durchgelaufen ist und ein Optimum gefunden hat, holt sich die Klasse *MemeticController* dieses Optimum aus dem *MemeticAlgorithm* und übergibt es an die Klasse *Whitebox*, die es zur Darstellung an die GUI weiterreicht. Desweiteren verwaltet die Klasse *Whitebox* Objekte vom Typ *TEvaluation*, die wichtig sind für das innerhalb des memetischen Algorithmus implementierte adaptive Verfahren. Diese Objekte werden vom *MemeticAlgorithm* in einem vector gespeichert und abgefragt über eine entsprechende Methode in der *Whitebox*.

Im nächsten Abschnitt folgen nun detailliertere Beschreibungen der einzelnen Klassen.

5.2.3. Klassenbeschreibungen für den memetischen Algorithmus

5.2.3.1. MemeticAlgorithm

Die Klasse *MemeticAlgorithm* realisiert den Ablauf des memetischen Algorithmus. Sie ist von der Klasse *EvolutionaryAlgorithm* abgeleitet, erbt also alle Methoden und Attribute des evolutionären Algorithmus. Die wichtigste öffentliche Methode ist die Methode *startCycle*, die aus *EvolutionaryAlgorithm* stammt und in *MemeticAlgorithm* überladen wurde, da im memetischen Algorithmus ja zusätzlich immer die lokale Suche aufgerufen werden muß. Außerdem werden innerhalb dieser Methode das adaptive Verfahren und die iterierte lokale Suche realisiert. Die Methode arbeitet im Groben wie folgt:
Als erstes wird die Initialisierung aufgerufen. Die erzeugten Individuen werden dann sofort durch einen Aufruf der Methode *localOptimization* lokal optimiert. Danach wird entweder die iterierte lokale Suche oder das normale memetische Verfahren ausgeführt.

Hat der Benutzer die iterierte lokale Suche ausgewählt, so wird nun je nach gewähltem Mutationsoperator entweder der Parameter *FLIPPINGPROBABILITY* oder der Parameter *NUMBEROFBITS* dynamisch in Abhängigkeit von der bereits gelaufenen Anzahl von Generationen angepaßt. Danach wird wie gewohnt die Mutation und anschließend die Methode *localOptimization* aufgerufen. Es wird weiterhin überprüft, ob das erzeugte Individuum ungültig geworden ist, also ob es noch innerhalb seiner Schranken liegt, die in der Klasse *Constraints* festgelegt wurden. Danach findet noch ein Aufruf der Umweltselektion statt, der die neue Elternpopulation für die nächste Generation erzeugt. Dieser Zyklus läuft solange, bis das angegeben Terminierungskriterium erfüllt ist. Am Ende wird eine Ergebnisdatei erstellt.

Hat der Benutzer sich für das normale memetische Verfahren entschieden, so wird zunächst die Elternselektion, dann die Rekombination, dann die Methode *localOptimization*, dann die Mutation und dann wieder die Methode *localOptimization* aufgerufen. Anschließend wird, wie bei der iterierten lokalen Suche auch, überprüft, ob die Individuen ungültig geworden sind. Anschließend wird die Population der besten Individuen aktualisiert und die Umweltselektion wird aufgerufen. Danach wird zusätzlich mit der Methode *testConvergence* überprüft, ob die Population konvergiert ist, d.h. ob zu viele Indiviuen das gleiche lokale Optimum darstellen. Ist dies der Fall, so wird die gesamte Population erneut mutiert und lokal optimiert. Hat der Benutzer zusätzlich das adaptive Verfahren aktiviert, so wird nach der zweiten Generation an dieser Stelle das adaptive

Verfahren angestoßen, indem sämtliche Punkte geholt werden, die das lokale Suchverfahren bisher angesehen hat. Es wird der insgesamt bisher beste Punkt ausgesucht und mit der Methode *calculateDistance* wird von jedem Punkt der euklidische Abstand zum besten Punkt berechnet und in einem vector-Objekt gespeichert. Das gleiche geschieht mit dem Fitneßwert jedes Individuums. Anschließend werden mit Hilfe der Methode *calculateStandardDeviation* die Standardabweichungen der Fitneß- und der Distanzwerte berechnet. Danach wird die Methode *calculateFDC* aufgerufen, die den Wert des Fitneß-Distanz-Korrelationskoeffizienten berechnet. Anhand dieses Wertes werden dann die Parameter für die Mutationsstärken angepaßt. Ganz am Ende des gesamten Zyklus wird eine Ergebnisdatei erstellt.

5.2.3.2. MemeticController

Die Klasse *MemeticController* stellt die Schnittstelle zwischen dem memetischen Algorithmus und der allgemeinen Verwaltungsklasse, der *Whitebox*, dar. Die wichtigste Methode im *MemeticController* ist die Methode *run*. In dieser Methode werden alle vom *MemeticAlgorithm* benötigten Objekte erzeugt und die Parametereinstellungen werden überprüft. Ist dies alles erfolgreich verlaufen, wird die Methode *startCycle* aus dem *MemeticAlgorithm* aufgerufen. Im folgenden wird nun etwas genauer beschrieben, was die Methode *run* tut und mit Hilfe welcher Methoden sie es tut. Das Erzeugen von Objekten und die Aufrufe der *check*-Methoden sehen für alle Objekte im wesentlichen gleich aus. Lediglich die Reihenfolge, in der sie erstellt werden, spielt eine Rolle.

Als erstes wird ein Objekt des memetischen Algorithmus erstellt. Dazu muß vorher überprüft werden, ob alle Parameter gesetzt wurden. Anschließend wird ein *RankingAndSelection*-Objekt erstellt, und zwar eines vom Typ *Means*, da dieser Typ des Ranking&Selection einen ganz normalen evolutionären Algorithmus simuliert. Als nächstes wird das *Constraints*-Objekt erzeugt. Auch hier werden zunächst mit Hilfe der *check*-Methode alle benötigten Parameter überprüft. Sobald einer der Paramter nicht korrekt gesetzt ist, wird eine *MAException* geworfen, die aber innerhalb der Klasse *MemeticController* wieder gefangen wird, und eine Fehlermeldung wird ausgegeben, die in einem log-Fenster in der GUI angezeigt werden soll. Wird eine *MAException* geworfen, so werden sofort alle bereits erstellten Objekt in umgekehrter Reihenfolge ihres Erstellens wieder gelöscht. Dieses Vorgehen wurde bei allen Überprüfungen von Parametern realisiert. Die nächsten Objekte, die erstellt werden, sind die Individuen. Außerdem werden Objekte der Elternpopulation, die Kindpopulation und die Elitepopulation erzeugt. Auch hier werden mittels *check*-Methoden die Parameter überprüft. Im weiteren Verlauf werden ein Initialisierungsobjekt, ein Objekt für die Elterselektion, ein Rekombinationsobjekt, ein Mutationsobjekt, ein Objekt für die Umweltselektion und ein Objekt für das Terminierungskriterium angelegt. Wurden alle Objekte erfolgreich erstellt, so wird die Methode *startCycle* in der Klasse *MemeticAlgorithm* aufgerufen. Ist diese Methode erfolgreich beendet, so wird aus dem *MemeticAlgorithm* mit Hilfe der Methode *getBestIndividual* das Optimum geholt und zurückgegeben. Vorher werden aber noch alle erstellten Objekte in umgekehrter Reihenfolge ihres Erstellens gelöscht. Die einzige Methode, die sonst noch in der Klasse *MemeticController* vorhanden ist, ist die Methode *evaluate*. Mit Hilfe die-

ser Methode wird eine gleich genannte Methode in der *Whitebox* aufgerufen, die eine Auswertung des übergebenen Punktes anstößt.

5.3. Leistungsstudien

Katharina Balzer

Dieses Kapitel beschäftigt sich mit den Leistungsstudien, die mit dem memetischen Algorithmus durchgeführt wurden. Die Funktionen, die als Optimierungsprobleme dienten, waren Schwefels Ridge, Ackley und Rastrigin (siehe Kapitel 2.4).

5.3.1. Untersuchungen ohne Rauschen

Im Fall ohne Rauschen wurden folgende Untersuchungen gemacht:

- Finde eine gute Konfiguration für den memetischen Algorithmus.
- Wie müssen die Mutationsstärken im adaptiven Verfahren, abhängig vom FDC, angepasst werden?
- Vergleiche den memetischen Algorithmus mit und ohne das adaptive Verfahren.
- Vergleiche den memetischen Algorithmus mit einem evolutionären Algorithmus mit gleicher Konfiguration.
- Vergleiche einen evolutionären Algorithmus mit InitRSM (Beschreibung siehe Kapitel 4.3.5.4) mit einem evolutionären Algorithmus mit uniformer Initialisierung.

Die Ergebnisse der Untersuchungen werden im folgenden erläutert.

Als gute Konfiguration für den memetischen Algorithmus ergab sich für Dimension 4 eine diskrete Rekombination (*ESDiscreteRecombination*), für Dimension 6 eine intermediäre Rekombination (*ESReco2Intermediate*). Die andere intermediäre Rekombination (*ESIntermediateRecombination*) stellte sich für alle Dimensionen als gut heraus. Als beste Umweltselektion ergab sich für alle Dimensionen die deterministische Selektion (*DeterministicSelection*). Bei den Mutationen gab es keinen Unterschied bei den Operatoren, lediglich die Mutationsstärken spielten eine größere Rolle. Bei der Standardbitmutation sollte eine Mutationsstärke von $\frac{1}{n}$ oder $\frac{2}{n}$ eingestellt werden, wobei n die Anzahl der Bits pro Koordinate multipliziert mit der Dimension darstellt. Bei der Bitmutation erwies sich eine Mutationsstärke von 1 oder 2 als optimal.

Dieses Ergebnis über die Mutationsstärken hatte dann auch direkten Einfluß auf das adaptive Verfahren. Es sollte ja untersucht werden, bei welchem FDC-Wert welche Mutationsstärken gut sind. Da aber immer kleine Mutationsstärken gut sind, ergab sich keinerlei Abhängigkeit vom FDC. Deshalb wurde die Anpassung so gestaltet, daß bei einem FDC-Wert zwischen 0.15 und 1.0 die optimalen Mutationsstärken eingestellt werden und bei einem FDC-Wert von -1.0 bis -0.15 werden die schlechtest möglichen Mutationsstärken eingestellt, d.h. bei Standardbitmutation eine Mutationsstärke von 0.5 und bei der Bitmutation eine von $\frac{n}{2}$. Diese Einstellung ergibt sich aus der Überlegung, daß bei

negativen FDC-Werten eine schlechte Relation zwischen der Distanz und dem Fitneßwert besteht, der Fitneßwert also mit steigernder Distanz auch steigt. Deshalb scheint es besser zu sein, in einem solchen Fall große Mutationsstärken zu benutzen. Diese Einstellung ist jedoch rein spekulativ, da wir keine Funktion vorliegen hatten, bei der sich ein negativer FDC-Wert ergab.

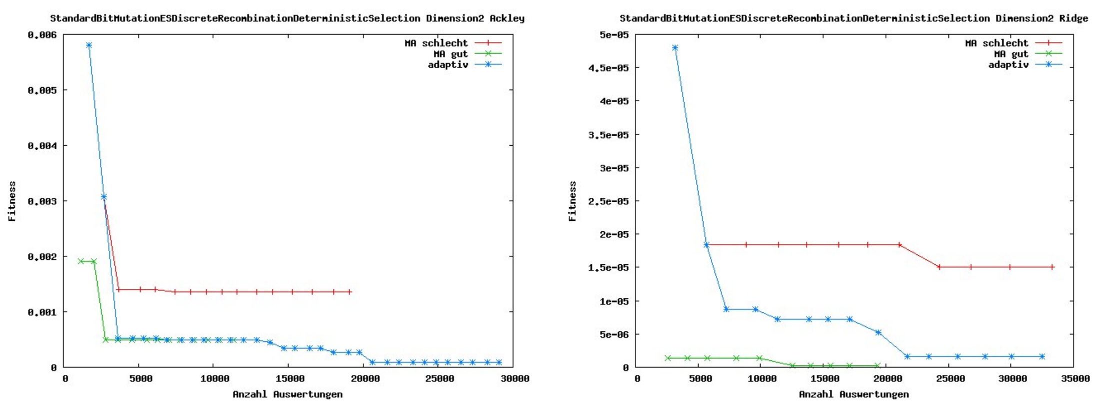


Abb. 5.3.: Vergleich zwischen gutem MA, schlechtem MA und adaptivem Verfahren

Dementsprechend künstlich fiel auch der Vergleich zwischen dem memetischen Algorithmus mit und dem ohne das adaptive Verfahren aus. Um überhaupt einen Effekt durch das adaptive Verfahren bekommen zu können, mußte das adaptive Verfahren mit schlechten Mutationsstärken gestartet werden. Verglichen haben wir diese Variante mit einem memetischen Algorithmus mit guter und einem mit schlechter Konfiguration. Das Ergebnis war, daß das adaptive Verfahren in seiner Güte zwischen den beiden anderen Verfahren liegt. Unschlagbar ist da natürlich der memetische Algorithmus mit guter Konfiguration. Beispiele für dieses Ergebnis sind in Abbildung 5.3 zu sehen.

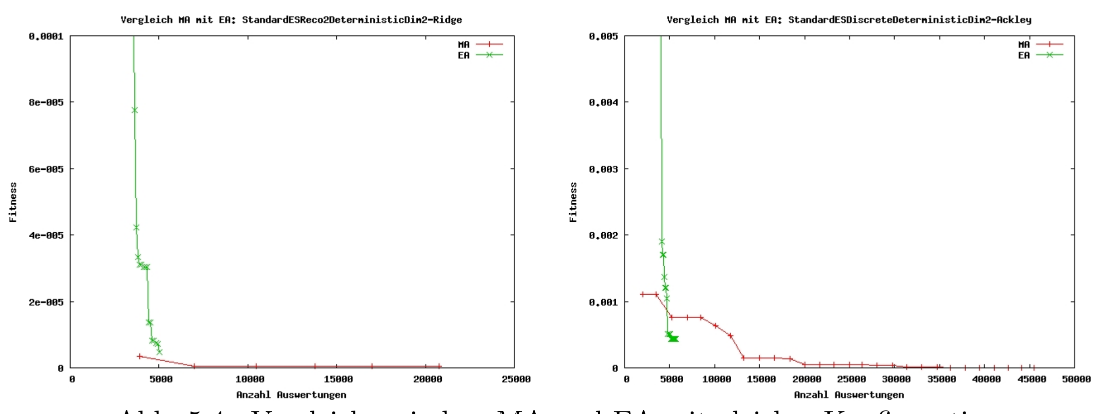


Abb. 5.4.: Vergleich zwischen MA und EA mit gleicher Konfiguration

Im Vergleich von memetischem Algorithmus und evolutionären Algorithmus ergab sich, daß der evolutionäre Algorithmus natürlich wie erwartet wesentlich weniger Auswertungen macht. Allerdings benötigt der memetische Algorithmus deutlich weniger Generationen, um ein besseres Individuum zu finden (siehe Abbildung 5.4).

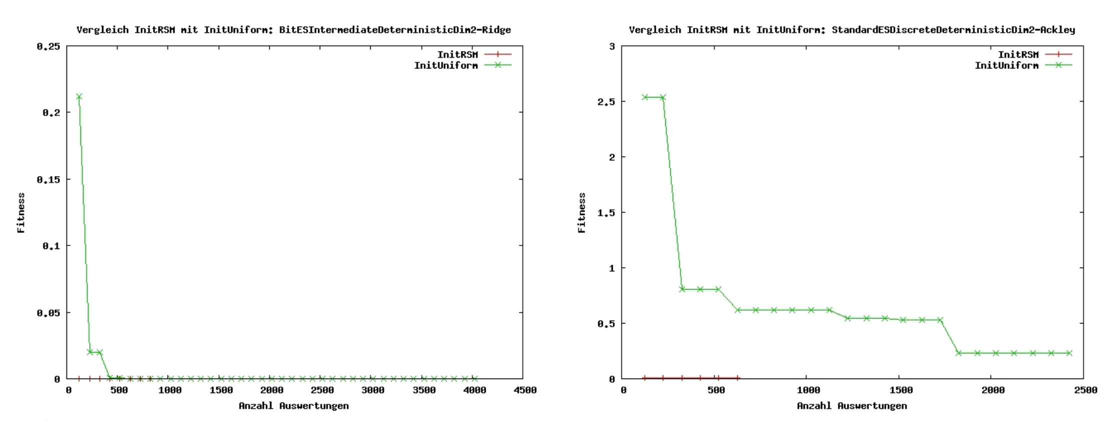


Abb. 5.5.: Vergleich zwischen Initialisierung mit RSM und uniformer Initialisierung

Der Vergleich eines evolutionären Algorithmus mit InitRSM als Initialisierung mit einem evolutionären Algorithmus mit uniformer Intitialisierung führte zu dem Ergebnis, daß bei Benutzung von InitRSM die Anfangspopulation schon erheblich besser war, als bei uniformer Initialisierung. Außerdem fand der EA mit InitRSM bei sonst gleicher Konfiguration und weniger Auswertungen ein besseres Ergebnis. D.h. der positive Effekt einer lokalen Optimierung direkt nach der Initialisierung läßt sich deutlich erkennen. Auch hierzu wurden Plots erstellt. Zwei davon sind in Abbildung 5.5 zu sehen.

5.3.2. Untersuchungen mit Rauschen

Im Fall mit Rauschen wurde der normale memetische Algorithmus in einer festen Konfiguration auf den in der Einleitung genannten Funktionen und verschiedenen Rauschstärken bei relativem Rauschen laufen gelassen. Die untersuchten Varianzen für das Rauschen waren 0.2, 0.4, 0.6, 0.8 und 1.0. Das Ergebnis dieser Untersuchungen war, daß der memetische Algorithmus durchaus robust ist gegenüber Rauschen. Plots zu diesen Untersuchungen sind in Abbildung 5.6 zu sehen.

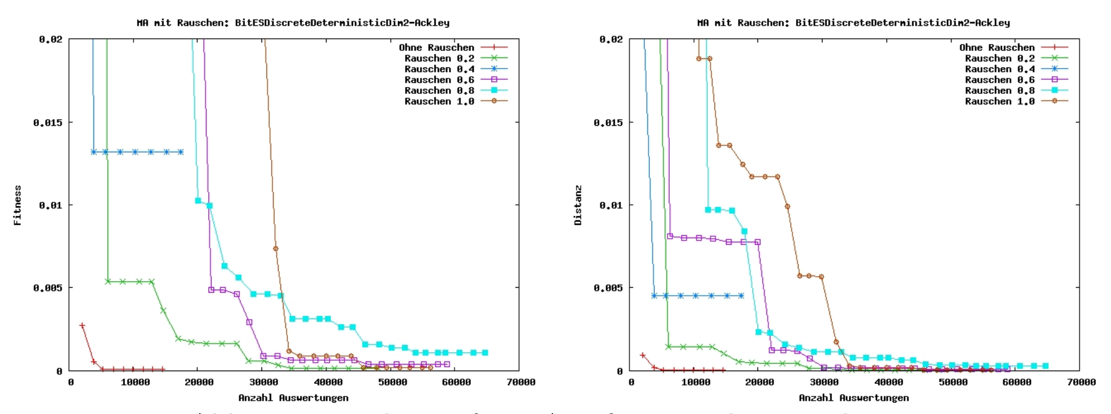


Abb. 5.6.: Ergebnisse für MA auf verrauschten Funktionen

6. Statistische Ranking & Selection Verfahren

6.1. Einleitung

Christian Horoba

Dieser Abschnitt informiert über zahlreiche statistische Auswahlverfahren (Ranking & Selection) und gibt Auskunft über deren Nutzen in Verbindung mit evolutionären Algorithmen bei der Optimierung verrauschter Funktionen.

In dem Unterabschnitt „Beschreibungen der Verfahren“ werden die genannten Verfahren kurz – aber vollständig – beschrieben. Im anschließenden Unterabschnitt „Klassenbeschreibungen“ wird auf die Implementierung der einzelnen Verfahren eingegangen. Der nachfolgende Unterabschnitt „Leistungsstudien“ stellt die Untersuchungsergebnisse dar, die mit einer Evolutionsstrategie mit integrierten statistischen Auswahlverfahren bei der Optimierung unterschiedlicher Funktionen gesammelt wurden.

6.2. Beschreibungen der Verfahren

6.2.1. Einleitung

Christian Horoba

In dieser Einleitung wird das Szenario, in dem statistische Auswahlverfahren Anwendung finden, vorgestellt.

Statistische Auswahlverfahren behandeln das allgemeine Problem, aus einer Menge von Lösungen eine beste Lösung auszuwählen, wobei den genannten Lösungen mithilfe einer vorliegenden Zielfunktion, die nur punktweise ausgewertet werden kann, jeweils eine reellwertige Güte zugeordnet wird. Falls es sich um eine deterministische Zielfunktion handelt, wird die Güte einer Lösung durch ihren Funktionswert quantifiziert. Falls es sich um eine probabilistische Zielfunktionen handelt, muß dieser Funktionswert nicht eindeutig bestimmt sein. Um daraus eine eindeutige reellwertige Größe, die die in diesem Kontext wesentliche Information enthält, zu extrahieren, betrachtet man deren Erwartungswert. Man kann die Güte einer Lösung folglich allgemein mit dem Erwartungswert einer Zufallsgröße identifizieren.

Ein Teil der Auswahlverfahren wählt keine einzelne Lösung, sondern eine Teilmenge der Lösungsmenge aus. Diese Teilmenge sollte im Idealfall aus besten Lösungen bestehen oder zumindest eine beste Lösung enthalten.

Im Fall deterministischer Zielfunktionen ist die oben vorgestellte Aufgabe trivial, da eine Funktionsauswertung genügt, um den Erwartungswert der eingeführten Zufallsgröße

zu bestimmen. Nach der einfachen Auswertung aller Lösungen ist man folglich in der Lage, diese zu sortieren und anschließend eine beste Lösung zu selektieren.

Im Fall probabilistischer Zielfunktionen stellt sich das Problem komplizierter dar, da im Allgemeinen keine Möglichkeit besteht, die Erwartungswerte der erwähnten Zufallsgrößen zu bestimmen. Diese können nur anhand von Realisationen der entsprechenden Zufallsgrößen geschätzt werden. Realisationen entsprechen hier den Funktionswerten einzelner – im Allgemeinen unabhängiger – Funktionsauswertungen.

Da die Bezeichnungen n_0, d und P in den folgenden Ausführungen häufig vorkommen, wird deren Bedeutung hier kurz angeschnitten. Der Parameter n_0 bezeichnet im Allgemeinen eine Anzahl von Funktionsauswertungen. d bezeichnet den Gleichheitsbereich-Parameter (indifference-zone parameter). Zwei Lösungen werden als im Wesentlichen gleichwertig betrachtet, wenn sich deren Güten um weniger als d unterscheiden. P bezeichnet im Allgemeinen eine Wahrscheinlichkeitsschranke für eine korrekte Auswahl des Selektionsverfahrens.

Im Folgenden werden einige Auswahlverfahren vorgestellt, die im geschilderten Szenario anwendbar sind.

6.2.2. Mittelwert-Auswahlverfahren

Christian Horoba

Das einfachste denkbare Verfahren, um aus einer Menge von Konfigurationen eine beste auszuwählen, besteht darin, jede Konfiguration n_0-mal auszuwerten, den Mittelwert zu berechnen und danach eine Konfiguration mit dem kleinsten Mittelwert zu wählen. Das Verfahren ist sehr einfach zu implementieren, verfügt aber auch über einige Unzulänglichkeiten: Es können neben dem Parameter n_0 keine weiteren Einstellungen vorgenommen werden; beispielsweise kann man keine Schranke für die Wahrscheinlichkeit einer richtigen Auswahl oder einen Gleichheitsbereich-Parameter vorgeben.

6.2.3. Konfidenzintervall-Auswahlverfahren

Christian Horoba

Eine weitere Möglichkeit, das Auswahlproblem zu behandeln, besteht darin, eine Wahrscheinlichkeit P und einen Parameter d vorzugeben und jede Konfiguration solange auszuwerten bis die Größe des entsprechenden P-Konfidenzintervalls kleiner als d ist. Anschließend kann eine beste Konfiguration anhand der Mittelwerte ermittelt werden. Diese Vorgehensweise hat den Vorteil, daß die Rauschstärke der betrachteten Funktion automatisch in die Berechnungen der Auswertungsanzahlen eingeht.

6.2.4. Enhanced Two-Stage Selection procedure (ETSS)

Peter Kissmann

Die Enhanced Two-Stage Selection Procedure (kurz: ETSS, siehe [8]) ist ein zwei-stufiges Ranking & Selection-Verfahren. In der ersten Stufe wird zunächst eine festgelegte Anzahl initialer Auswertungen durchgeführt. Auf Basis dieser Auswertungen läßt sich aufgrund der Varianz-Schätzungen bestimmen, wie viele Auswertungen in der zweiten Stufe für

die einzelnen Konfigurationen durchgeführt werden sollen. Hierbei berechnet sich die Anzahl der Auswertungen für Konfiguration i für die zweite Stufe wie folgt:

$$N_i = \max\left(n_0, \left\lceil \left(\frac{hS_i(n_0)}{d^*}\right)^2 \right\rceil\right), \qquad \text{für } i = 1, 2, \ldots, k \tag{6.1}$$

h ist hierbei eine Konstante, die Rinotts Integral läßt. Vertafelte Werte für h lassen sich beispielsweise in [23] finden.

Es ist jedoch vorteilhaft, offensichtlich schlechte Konfigurationen nicht weiter auszuwerten. Dazu wird mittels einer Teilmengen-Selektions-Methode (siehe [9]) die Teilmenge der Konfigurationen bestimmt, die den anderen deutlich überlegen sind: Die Konfiguration i_l, mit der besten Mittelwertschätzung $\hat{\mu}_{i_l} = \min_{i=1,2,\ldots,k}(\hat{\mu}_i)$ wird automatisch in die Teilmenge der besten Konfigurationen aufgenommen. für alle $i \neq i_l$ bestimme

$$t = t_{1-(1-P^*)/(k-1),n_0-1}$$

(beachte: hier wurde die Formel aus [18], Anmerkung im Abschnitt 3, Seite 8, verwendet, das einen Vorgänger-Artikel von [8] darstellt; im letztgenannten wurde diese Formel leider falsch abgedruckt),

$$S^2_{i-i_l} = \frac{\sum_{j=1}^{n_0}\left(X_{ij} - X_{i_l j} - (\hat{\mu}_i - \hat{\mu}_{i_l})\right)^2}{n_0 - 1}$$

und

$$W_{i,i_l} = \frac{tS_{i,i_l}}{\sqrt{n_0}}.$$

Eine Konfiguration i wird nun zur Teilmenge hinzugefügt, wenn $\hat{\mu}_i - \hat{\mu}_{i_l} \leq (W_{i,i_l} - d^*)^+$ gilt. Durch die hier verwendete Art der Varianz-Schätzung ist dieses Verfahren optimal für die Benutzung von Common Random Numbers (siehe Abschnitt 2.3) geeignet.

Der gesamte Algorithmus sieht dann im Überblick folgendermaßen aus:

6.2.4.1. Der ETSS-Algorithmus

1. Führe n_0 Auswertungen durch.
2. Führe die Teilmengen-Selektion durch.
3. Wenn nur ein Element in der Teilmenge liegt, gehe zu Schritt 6.
4. Für jede Konfiguration i in der Teilmenge berechne die nötige Anzahl zusätzlicher Auswertungen $N_i - n_0$. N_i wird hierbei gemäß Gleichung 6.1 berechnet.
5. Simuliere diese $N_i - n_0$ zusätzlichen Auswertungen für jede Konfiguration i der Teilmenge.
6. Gib die den Index i_1 der Konfiguration mit der besten Mittelwertschätzung und die Mittelwertschätzung $\hat{\mu}_{i_1}$ zurück.

INPUT: Minimale Anzahl initialer Auswertungen n_0,
ungefähre Wahrscheinlichkeitsschranke für eine richtige Auswahl P_{app},
Gleichheitsbereich-Parameter d^*,
Menge $H = \{1, \dots, k\}$ mit k konkurrierenden Konfigurationen,
maximale Teilmengengröße m,
Anzahl vorhandener Auswertungen n_i für Konfiguration i, $i = 1, \dots, k$.
OUTPUT: Menge H mit maximal m beibehaltenen Konfigurationen.

(1) Berechne die angepaßte Wahrscheinlichkeitsschranke $P^* = P_{app}^{\frac{1}{k-m}}$.
(2) Berechne $n = \min(\{n_0\} \cup \{n_i : i = 1, \dots, k\})$.
(3) WHILE $|H| > m$ DO
(4) FOR ALL $i \in H$ DO
(5) Berechne Auswertungen $X_{i1}, \dots, X_{in}$, falls noch nicht geschehen.
(6) OD
(7) Rufe *extended screen-to-the-best*-Prozedur mit H, P^*, $\frac{d^*}{2}$ und n_i auf.
(8) $n := n + 1$.
(9) OD

Abb. 6.1.: *Iterative subset selection*-Prozedur.

6.2.5. Iterative subset selection (ISS)

Christian Horoba

Bei dem in [7] vorgestellten Auswahlverfahren *iterative subset selection* handelt es sich um eine Heuristik, die aus einer gegebenen Konfigurationenmenge eine Teilmenge, die eine beste Konfiguration enthält, auswählt. Für die Größe dieser Teilmenge kann eine feste Schranke m vorgegeben werden. Solange die vorliegende Menge mehr als m Konfigurationen enthält, werden weitere Funktionsauswertungen angestoßen und im Anschluß die *extended screen-to-the-best*-Prozedur aufgerufen. Das Auswahlverfahren kann leider nicht die Einhaltung einer festen Wahrscheinlichkeitsschranke für eine richtige Auswahl sicherstellen.

Das kurz beschriebene Verfahren wird detailliert in Abbildung 6.1 dargestellt. Abbildung 6.2 zeigt die verwendete *extended screen-to-the-best*-Prozedur, die in [5] vorgeschlagen wurde.

Da die *extended screen-to-the-best*-Prozedur unabhängige Auswertungen der Konfigurationen voraussetzt, kann sie nicht zusammen mit gemeinsamen Zufallszahlen verwendet werden, da letztere im Allgemeinen zu einer positiven Korrelation der Auswertungen führen (siehe Abschnitt 2.3). In Abbildung 6.3 geben wir eine modifizierte Version der *extended screen-to-the-best*-Prozedur an, von der speziell in diesem Szenario bessere Ergebnisse zu erwarten sind.

INPUT: Wahrscheinlichkeitsschranke für eine richtigen Auswahl P^*,
Gleichheitsbereich-Parameter d^*,
Menge $H = \{1, \ldots, k\}$ mit k konkurrierenden Konfigurationen,
Anzahl vorhandener Auswertungen n_i, $i = 1, \ldots, k$.
OUTPUT: Menge H mit beibehaltenen Konfigurationen.
(1) FOR $i = 1, \ldots, k$ DO
(2) Berechne Erwartungswert- und Varianzschätzung:
$\bar{X}_i = \frac{1}{n_i} \sum_{j=1}^{n_i} X_{ij}, \qquad S_i^2 = \frac{1}{n_i - 1} \sum_{j=1}^{n_i} \left(X_{ij} - \bar{X}_i\right)^2.$
(3) Sei t_i das $(P^*)^{\frac{1}{k-1}}$-Quantil der t-Verteilung mit $n_i - 1$ Freiheitsgraden.
(4) OD
(5) FOR $i = 1, \ldots, k$ and $j = 1, \ldots, k$ DO
(6) Berechne Gewicht:
$W_{ij} = \sqrt{\frac{t_i^2 S_i^2}{n_i} + \frac{t_j^2 S_j^2}{n_j}}.$
(7) IF $-\bar{X}_i < -\bar{X}_j - \max\{0, W_{ij} - d^*\}$ THEN Entferne Konfiguration i aus H.
(8) OD

Abb. 6.2.: *Extended screen-to-the-best*-Prozedur.

INPUT: Wahrscheinlichkeitsschranke für eine richtigen Auswahl P^*,
Gleichheitsbereich-Parameter d^*,
Menge $H = \{1, \ldots, k\}$ mit k konkurrierenden Konfigurationen,
Anzahl vorhandener Auswertungen n_i, $i = 1, \ldots, k$.
OUTPUT: Menge H mit beibehaltenen Konfigurationen.
(1) FOR $i = 1, \ldots, k$ DO
(2) Berechne Erwartungswertschätzung:
$\bar{X}_i = \frac{1}{n_i} \sum_{j=1}^{n_i} X_{ij}.$
(4) OD
(5) FOR $i = 1, \ldots, k$ and $j = 1, \ldots, k$ DO
(6) Berechne $n_{ij} = \min\{n_i, n_j\}$.
(7) Sei t_{ij} das $(P^*)^{\frac{1}{k-1}}$-Quantil der t-Verteilung mit $n_{ij} - 1$ Freiheitsgraden.
(8) Berechne $S_{ij}^2 = \frac{1}{n_{ij} - 1} \sum_{k=1}^{n_{ij}} \left(X_{ik} - X_{jk} - \left(\bar{X}_i - \bar{X}_j\right)\right)^2$.
(9) Berechne Gewicht:
$W_{ij} = \frac{t_{ij} S_{ij}}{\sqrt{n_{ij}}}.$
(10) IF $-\bar{X}_i < -\bar{X}_j - \max\{0, W_{ij} - d^*\}$ THEN Entferne Konfiguration i aus H.
(11) OD

Abb. 6.3.: Für die Benutzung von gemeinsamen Zufallszahlen angepaßte *extended screen-to-the-best*-Prozedur.

6.2.6. Auswahlverfahren nach Koenig und Law

Christian Horoba

Das im Folgenden vorgestellte Verfahren ist [12] entnommen.

Seien $\Pi_1, \ldots, \Pi_k$, $k \geq 2$, normalverteilte Zufallsgrößen. Bezeichne μ_i den Erwartungswert der Zufallsgröße Π_i, $1 \leq i \leq k$, und sei $(r(1), \ldots, r(k))$ eine Permutation von $(1, \ldots, k)$, so daß $\mu_{r(1)} \leq \ldots \leq \mu_{r(k)}$ gilt. Seien δ^*, k, m, l und P^* so gewählt, daß $0 < \delta^*$, $1 \leq l \leq m < k$ und $\frac{m! \cdot (k-l)!}{(m-l)! \cdot k!} < P^* < 1$ gilt. Das zu behandelnde Auswahlproblem entspricht dann der zufälligen Auswahl einer Menge $M \subseteq \{\Pi_1, \ldots, \Pi_k\}$ von m Zufallsgrößen, so daß die Wahrscheinlichkeit, daß M mindestens l Zufallsgrößen, deren Erwartungswerte kleiner als $\mu_{r(l)} + \delta^*$ sind, enthält, mindestens P^* beträgt.

Nun wird ein zweistufiges Verfahren zur Lösung des formalisierten Problems beschrieben. Es wird angenommen, daß der Gleichheitsbereich-Parameter δ^* und die gewünschte Wahrscheinlichkeit P^* im Vorfeld festgelegt wurden. Nun werden initial n_0 Realisationen $x_i(1), \ldots, x_i(n_0)$ der Zufallsgröße Π_i, $1 \leq i \leq k$, rein zufällig gezogen. Seien

$$\overline{x_i}^{(1)} = \frac{1}{n_0} \cdot \sum_{j=1}^{n_0} x_i(j)$$

und

$$s_i^2 = \frac{1}{n_0 - 1} \cdot \sum_{j=1}^{n_0} \left[x_i(j) - \overline{x_i}^{(1)} \right]^2 .$$

Sei h die positive reelle Zahl, die die folgende Gleichung erfüllt:

$$(k-m) \cdot \binom{k-l}{m-l} \cdot \int_{-\infty}^{+\infty} [F(t+h)]^l \cdot [F(t)]^{m-l} \cdot [1 - F(t)]^{k-m-1} \cdot f(t) \, \mathrm{d}t = P^*,$$

wobei f und F die Dichte- bzw. Verteilungsfunktion der t-Verteilung mit $n_0 - 1$ Freiheitsgraden bezeichnen. Sei $z = \left(\frac{\delta^*}{h}\right)^2$. Die Gesamtanzahl von Funktionsauswertungen N_i, die für die Zufallsgröße Π_i, $1 \leq i \leq k$, benötigt werden, wird durch die folgende Gleichung bestimmt:

$$N_i = \max \left\{ n_0 + 1, \left\lceil \frac{s_i^2}{z} \right\rceil \right\},$$

wobei $\lceil y \rceil$ die kleinste ganze Zahl, die größer oder gleich y ist, bezeichnet. Als nächstes werden $N_i - n_0$ zusätzliche Auswertungen $x_i(n_0 + 1), \ldots, x_i(N_i)$ der Zufallsgröße Π_i, $1 \leq i \leq k$, vorgenommen und daraus der Mittelwert

$$\overline{x_i}^{(2)} = \frac{1}{N_i - n_0} \cdot \sum_{j=n_0+1}^{N_i} x_i(j)$$

der zweiten Stufe berechnet. Definiere die Gewichte

$$W_{i1} = \frac{n_0}{N_i} \cdot \left(1 + \left\{ 1 - \frac{N_i}{n_0} \cdot \left[1 - \frac{(N_i - n_0) \cdot z}{s_i^2} \right] \right\}^{\frac{1}{2}} \right)$$

und

$$W_{i2} = 1 - W_{i1}$$

für $1 \le i \le k$. Definiere abschließend den gewichteten Mittelwert

$$T_i = W_{i1} \cdot \overline{x_i}^{(1)} + W_{i2} \cdot \overline{x_i}^{(2)}.$$

Sei M die Menge der m Zufallsgrößen mit den kleinsten T_i's.

Man kann zeigen, daß die Menge M der ausgewählten Zufallsgrößen die folgende Eigenschaft besitzt:

$$\text{Prob}(M \text{ enthält mindestens } l \text{ Lösungen, deren Erwartungswerte kleiner als } \mu_{r(l)} + \delta^* \text{ sind}) \ge P^* .$$

Der formale Beweis dieser Aussage kann in [12] nachgelesen werden.

6.2.7. Sequential Selection with Memory (SSM)

Peter Kissmann

Auch in der Prozedur "Sequential Selection with Memory" (kurz SSM; siehe [20]) werden in einer ersten Stufe die vom Benutzer spezifizierten initialen Auswertungen durchgeführt. Nach diesen initialen Auswertungen wird bestimmt, welche Konfigurationen offensichtlich sehr schlecht sind, so daß sie nicht weiter ausgewertet werden müssen. Sollte hierdurch nur noch eine einzige Konfiguration überleben, wird diese als die Beste zurückgegeben und der Algorithmus stoppt. Anderenfalls werden die überlebenden Konfigurationen der Reihe nach je einmal ausgewertet und es wird wieder überprüft, welche Konfigurationen erheblich schlechter als andere sind. Diese Schritte werden so lange wiederholt, bis entweder nur noch eine Konfiguration überlebt oder eine zuvor berechnete maximale Anzahl an Durchläufen durchgeführt wurde. In jedem Fall stoppt der Algorithmus dann und gibt die Konfiguration mit der besten Mittelwertschätzung zurück.

Im Einzelnen läuft die Prozedur wie folgt ab:

6.2.7.1. Die SSM-Prozedur

1. *Initialisierung:* Führe die initialen Auswertungen durch, um die Varianz σ_{ij}^2 zu schätzen.

   - Erzeuge zwei Mengen V und V^c, die die Indizes der Konfigurationen enthalten, die bereits ausgewertet wurden (V) beziehungsweise die noch keinen Auswertungen unterzogen wurden (V^c).
   - Für jede Konfiguration π_i, $i \in V^c$, führe $n_0 \ge 2$ Beobachtungen durch und setze $n_i = n_0$.
   - Für $i \in V$ führe, falls $n_i < n_0$, zusätzliche Beobachtungen durch und aktualisiere V und V^c wie folgt:

$$\begin{aligned} V^c &= V^c \cup \{i\} \\ V &= V \setminus \{i\} \end{aligned}$$

Berechne die Varianz-Schätzung

$$S_{ij}^2 = \frac{1}{n_0 - 1} \sum_{p=1}^{n_0} \left(X_{ip} - X_{jp} - \left[\bar{X}_i (n_0) - \bar{X}_j (n_0) \right] \right)^2 .$$

2. *Prozedur-Parameter:* Sei c eine positive ganze Zahl. Berechne λ und a_{ij} wie folgt:

$$\lambda = \frac{\delta}{2c} \qquad \text{und} \qquad a_{ij} = \frac{\eta f S_{ij}^2}{4(\delta - \lambda)},$$

wobei η folgende Gleichung erfüllt:

$$\sum_{p=1}^{c} (-1)^{p+1} \left(1 - \frac{1}{2} I(p = c) \right) .$$

$$\left(1 + \frac{(2c-p)\, p\eta}{2c-1} \right)^{-f/2} = \frac{\alpha}{k-1}.$$

$I(\cdot)$ repräsentiert hierbei die Indikatorfunktion. Die letzte Gleichung hat für $c = 1$ eine Lösung in geschlossener Form:

$$\lambda = \frac{\delta}{2} \qquad \text{und} \qquad a_{ij} = \frac{f S_{ij}^2}{4(\delta - \lambda)} \left[\left(\frac{k-1}{2\alpha} \right)^{2/f} - 1 \right].$$

Aus diesen Werten berechne die maximale Anzahl Auswertungen N mittels

$$\begin{aligned} N_{ij} &= \left\lfloor \frac{a_{ij}}{\lambda} \right\rfloor \\ N_i &= \max_{j \neq i} \{N_{ij}\} \\ N &= \max_i N_i. \end{aligned}$$

Ist nun $n_0 > N$, so stoppe und wähle die Lösung mit der kleinsten Erwartungswertschätzung

$$\bar{X}_i(n_i) = \frac{1}{n_i} \sum_{p=1}^{n_i} X_{ip}$$

als die beste. Anderenfalls sei $I = \{1, 2, \ldots, k\}$ die Menge der überlebenden Lösungen. Setze $r = n_0$ und fahre beim Schritt *Screening* fort. Von nun an repräsentiert V die Menge der Lösungen, auf denen mehr als r Auswertungen durchgeführt wurden, während V^c die Menge der Lösungen repräsentiert, die exakt r-mal ausgewertet wurden.

3. *Screening:* Setze $I^{alt} = I$. Sei

$$I = \left\{ i : i \in I^{alt} \text{ und } R_i \geq \max_{j \in I^{alt}, j \neq i} (R_j - a_{ij}) + r\lambda \right\},$$

mit

$$R_j = \begin{cases} -\sum_{p=1}^{r} X_{jp} & \text{für } j \in V^c \\ -r\bar{X}_j(n_j) & \text{für } j \in V. \end{cases}$$

4. *Stopp-Regel:* Falls $|I| = 1$, so stoppe und gib die einzige überlebende Konfiguration als die beste zurück; anderenfalls führe eine weiter Auswertung für jedes Konfiguration π_i, $i \in (I \cap V^c)$, durch und setze $r = r + 1$. Falls $r = N + 1$, so terminiere die Prozedur und wähle die Lösung aus I, die die niedrigste Mittelwertschätzung hat, als die beste aus; anderenfalls setze für alle $i \in (I \cap V)$ mit $n_i = r$

$$\begin{aligned} V^c &= V^c \cup \{i\} \\ V &= V \setminus \{i\} \end{aligned}$$

und gehe weiter zum Schritt *Screening*.

6.3. Klassenbeschreibungen

Peter Kissmann

6.3.1. Allgemeine Implementierungsdetails

Die wichtigste Anforderung an die Implementierung der Ranking & Selection-Verfahren war es, eine Schnittstelle zu schaffen, die von allen Algorithmen genutzt werden konnte. Um dies zu erzielen, wurde die neue Klasse *Configuration* erstellt. Diese schachtelt die wichtigsten Informationen über eine Konfiguration, welche die Position im Suchraum sowie die bisher auf dieser Konfiguration durchgeführten Auswertungen sind. Zur Speicherung der Auswertungen wurde die Datenstruktur einer verketteten Liste gewählt, da diese keine Längenbeschränkung (im Gegensatz zu Feldern) hat und sich durch Zeiger auf Kopf- und Schlußelement effizient erweitern läßt. Darüberhinaus bietet die *Configuration*-Klasse die Möglichkeit, weiterführende Informationen zurückzugeben. So ist es hier beispielsweise möglich, Erwartungswertschätzungen, Varianzschätzungen und Konfidenzintervallbreiten der gespeicherten Auswertungen abzufragen.

Um nun die einzelnen Ranking & Selection-Verfahren durchführen zu können, wurde eine abstrakte Klasse (*RankingAndSelection*) erstellt. Sie verfügt über eine abstrakte Methode (*determineBestConfigurations*), die sich um das eigentlich Ranking & Selection kümmert und somit von den einzelnen Verfahren implementiert werden muß. Für die Auswertungen, die in der *BlackBox* angestoßen werden, steht die Methode *evaluate* zur Verfügung, die schon vollständig implementiert ist, da die Auswertung unabhängig von den einzelnen Verfahren ist. Durch diese allgemeine Oberklasse ist es problemlos möglich, weitere R&S-Operatoren zu implementieren und in die Algorithmen einzubinden, was auch zwischenzeitlich getan wurde - beispielsweise als sich herausstellte, daß ISS nicht für Common Random Numbers geeignet ist und somit die Klasse *ISS_CRN* nachträglich noch implementiert wurde.

Im Klassendiagramm (6.4) sind alle für die Ranking & Selection-Verfahren implementierten Klassen abgebildet. Diese werden in den folgenden Abschnitten detaillierter beschrieben.

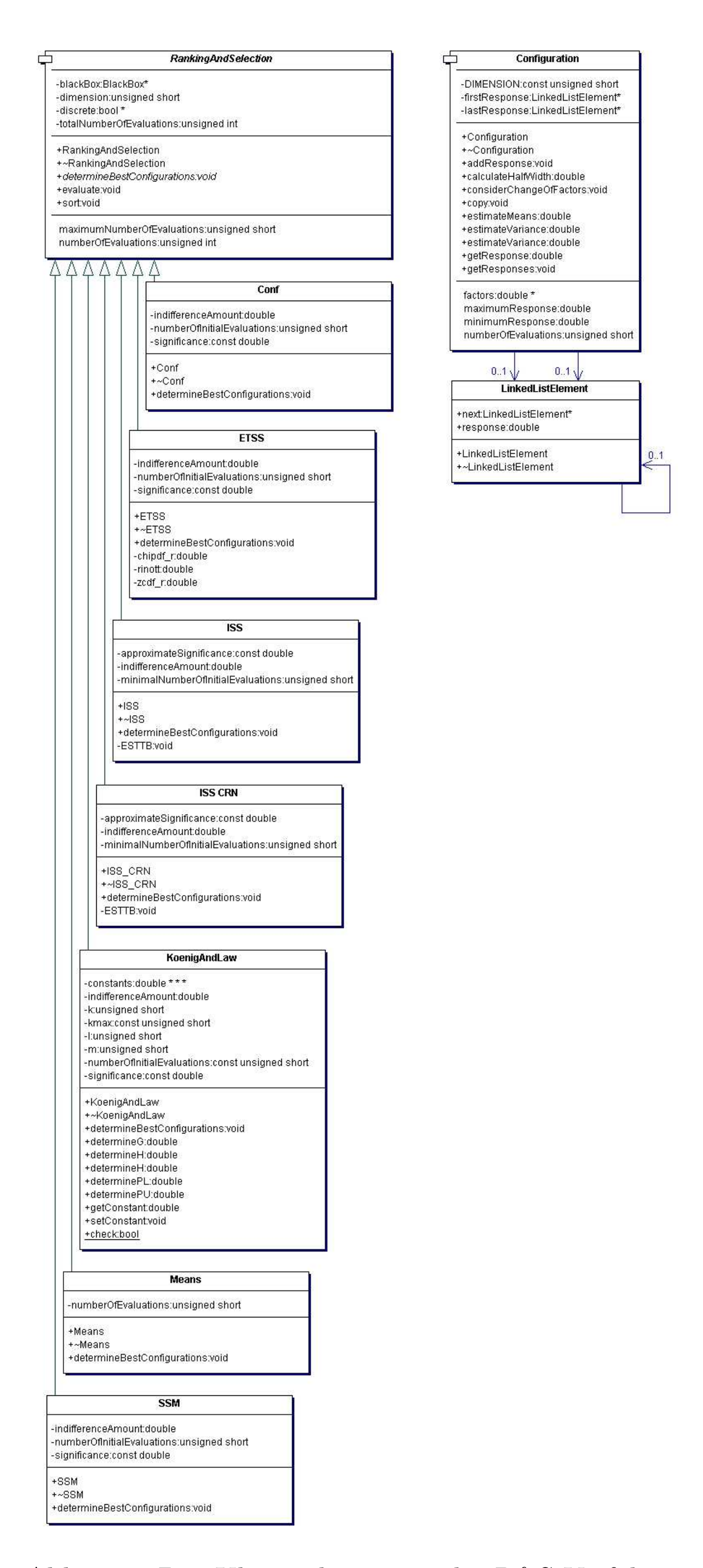


Abb. 6.4.: Das Klassendiagramm der R&S-Verfahren

6.3.2. Configuration

Die Klasse *Configuration* kapselt alle Informationen, die über eine Konfiguration bekannt sind. Im Konstruktor werden ihr die Faktorbelegungen als Feld sowie die Dimension (damit später auf die einzelnen Faktoren gültig zugegriffen werden kann) übergeben. Abgesehen davon, diese Dinge zu speichern, setzt er die Zeiger auf das erste und letzte Element der intern verwalteten Liste von Auswertungen auf $NULL$ und initialisiert auch die Anzahl von Auswertungen mit 0.

Sobald auf einer Konfiguration eine Auswertung durchgeführt wird, sollte sie mittels *addResponse* hinzugefügt werden. In diesem Fall wird ein neues *LinkedListElement* (siehe Abschnitt 6.3.2.1) erzeugt, in dem der Response-Wert der Auswertung gespeichert wird, und am Ende der Liste angehängt. Zusätzlich wird die Anzahl der Auswertungen inkrementiert, so daß dieser Zähler stets aktuell ist und die Länge der Liste angibt.

Diese Anzahl an Auswertungen läßt sich einfach mittels *getNumberOfEvaluations* anfragen. Auch die Faktoren können auf ähnliche Weise durch *getFactors* geholt werden. Sobald Auswertungen auf einer Konfiguration durchgeführt wurden, lassen sich auch diese entweder einzeln (*getResponse*) durch Übergabe des Index der entsprechenden Auswertung oder als Feld über einen größeren Bereich (*getResponses*) zurückliefern. Im letzteren Fall müssen die Indizes für die erste und die letzte Auswertung, die zurückgegeben werden sollen, sowie ein Feld, in das diese dann geschrieben werden, übergeben werden. Werden keine Indizes übergeben, so werden alle Auswertungen zurückgegeben. Desweiteren ist es möglich, die minimale sowie die maximale Response (mittels *getMinimumResponse* beziehungsweise *getMaximumResponse*) abzufragen.

Auf Basis der durchgeführten Auswertungen ist es möglich, den Mittelwert oder die Varianz schätzen zu lassen sowie die Breite des Konfidenzintervalles zu berechnen. Für die Mittelwertschätzung steht die Methode *estimateMeans* zur Verfügung. Ihr werden der Index der ersten und der Index der letzten Auswertung übergeben, über die der Mittelwert gebildet werden soll. Werden keine Indizes übergeben, so wird der Mittelwert aller Auswertungen zurückgegeben. Für die Varianzschätzung wurde die Methode *estimateVariance* in zwei Ausprägungen implementiert: Zum einen eine Version, der, analog zu *estimateMeans*, die Indizes der ersten und letzten Auswertung übergeben werden, von denen die Varianz geschätzt werden soll. Zum anderen eine Version, der neben den beiden Indizes, zusätzlich noch die Mittelwertschätzung übergeben wird, die zur Bestimmung der Varianz benötigt wird. Um die halbe Breite des Konfidenzintervalles bestimmen zu lassen, muß die Methode *calculateHalfWidth* aufgerufen werden und dieser zum einen eine untere Wahrscheinlichkeitsschranke dafür, daß das Intervall den Erwartungswert enthält, zum anderen die Art, wie dieses Intervall berechnet werden soll, übergeben werden. Hierfür stehen die Werte 0, 1 und 2 zur Auswahl. 0 bedeutet, daß das Intervall durch Nutzung der T-Verteilung bestimmt werden soll; bei 1 wird die Normalverteilung genommen; und bei 2 wird das Intervall durch Verwendung der Tschebyscheffschen Ungleichung berechnet.

Für den Fall, daß sich an den Faktoren etwas ändert - in diesem Fall ergibt sich natürlich eine neue Konfiguration - wurde die Methode *considerChangeOfFactors* implementiert. Diese sorgt dafür, daß alle Auswertungen, die bisher gespeichert wurden,

gelöscht werden sowie die Anzahl an Auswertungen auf 0 zurückgesetzt wird. Das hat den Vorteil, daß nicht jedes Mal, wenn eine Faktoränderung anfällt, das alte *Configuration*-Objekt gelöscht und ein völlig neues erstellt werden muß.

Als letzte Methode gibt es noch eine einfache *copy*-Methode, die zunächst die Faktoren der übergebenen Konfiguration kopiert und anschließend *considerChangeOfFactors* aufruft, um die eigenen alten Auswertungen komplett zu löschen. Sobald dies geschehen ist, werden alle Auswertungen der übergebenen Konfiguration kopiert und der Zähler der Anzahl an Auswertungen aktuell gehalten.

6.3.2.1. LinkedListElement

Die Klasse *LinkedListElement* stellt die Datenstruktur einer verketteten Liste zur Verfügung und dient hier dazu, eine Liste von Auswertungen einer Konfiguration zu speichern. Dazu verfügt sie über einen Zeiger auf das nächste Element (*next*) sowie das Attribut *response*, das den Wert einer Auswertung darstellt. Da diese Attribute öffentlich sind, ist es möglich, auch von außen auf diese zuzugreifen und es wurde auf jegliche get- und set-Methoden verzichtet.

6.3.3. RankingAndSelection

Die Klasse *RankingAndSelection* dient, wie schon erwähnt, als gemeinsame Oberklasse aller Ranking & Selection-Verfahren und wurde deshalb als abstrakte Klasse erstellt.

Dem Konstruktor muß ein Zeiger auf die *BlackBox* übergeben werden, der auch gespeichert wird, um die darin enthaltene *Evaluate*-Methode aufrufen zu können.

Für die Durchführung des eigentlichen R&S-Verfahrens wurde die abstrakte Methode *determineBestConfigurations* erstellt. Diese muß von den einzelnen Verfahren implementiert werden. Übergeben werden ihr ein Feld von Indizes, die zurückgeliefert werden sollen sowie dessen Länge. Wenn das Verfahren durchgelaufen ist, bestimmt es die besten Konfigurationen und schreibt deren Indizes in das übergebene Feld. Desweiteren müssen die Konfigurationen, die verglichen werden sollen (ebenfalls als Feld), sowie deren Anzahl übergeben werden.

Die einzelnen Verfahren müssen unter Umständen neue Auswertungen anstoßen. Hierfür steht die Methode *evaluate* zur Verfügung, der ein Zeiger auf eine Konfiguration übergeben werden muß. Sie holt sich dann die Faktoren der Konfiguration, rundet diejenigen Faktoren, die diskret sein sollen, und ruft hiermit dann die *Evaluate*-Methode der *BlackBox* auf. Das Ergebnis der Auswertung wird in die Konfiguration geschrieben und ein interner Zähler für die Anzahl an Auswertungen, die mit diesem R&S-Verfahren durchgeführt wurden, inkrementiert. Anschließend wird noch geschaut, ob auch die maximale Anzahl an Auswertungen, die für eine Konfiguration berechnet wurden, durch die neue Auswertung geändert wurde und das Attribut gegebenenfalls aktualisiert. Diese beiden Attribute lassen sich mittels *getNumberOfEvaluations* beziehungsweise *getMaximumNumberOfEvaluations* abfragen.

Schlußendlich wurde noch eine *sort*-Methode geschrieben, die ein übergebenes Indizes-Feld anhand der korrespondierenden Werte, die in einem ebenfalls übergebenen Feld

gespeichert sind, sortiert. Zusätzlich muß natürlich noch die Länge der beiden Felder übergeben werden. Die einzelnen Verfahren sortieren (innerhalb der Methode *determine-BestConfigurations*) nach der Auswertung die Indizes der Konfigurationen typischerweise anhand der Mittelwertschätzungen (beim Verfahren nach Koenig und Law ist dies ein gewichteter Mittelwert) und wählen die gewünschte Anzahl von Indizes aus, die zurückgegeben werden sollen. Diese stehen normalerweise nach dem Sortieren auf den ersten Positionen des Index-Feldes.

6.3.3.1. Conf

Die Klasse *Conf* implementiert das R&S-Verfahren, das die Konfigurationen so lange auswertet, bis die Breite des Konfidenzintervalles kleiner als der übergebene *indifferenceAmount* ist (siehe Abschnitt 6.2.3). Dazu werden im Konstruktor neben der *Black-Box* noch besagter *indifferenceAmount* sowie die Anzahl initialer Auswertungen und die Schranke für das Konfidenzintervall übergeben.

Die Methode *determineBestConfigurations* implementiert das eigentliche Verfahren. Nach den nötigen Auswertungen werden die Indizes der Konfigurationen anhand der Mittelwertschätzung aller Auswertungen der einzelnen Konfigurationen sortiert und die gewünschte Anzahl bester Indizes zurückgegeben.

6.3.3.2. ETSS

Die Klasse *ETSS* implementiert die *Enhanced Two-Stage Selection*-Prozedur, die in Abschnitt 6.2.4 beschrieben wurde. Auch hier müssen im Konstruktor die *BlackBox*, der *indifferenceAmount*, die Anzahl Auswertungen für die erste Stufe sowie die Schranke für die Wahrscheinlichkeit einer korrekten Auswahl übergeben werden.

Neben der *determineBestConfigurations*-Methode, die alle Verfahren implementieren müssen und in der das eigentliche Verfahren durchgeführt wird, wurden noch drei private Methoden eingefügt, die dazu dienen, das Rinott-Integral zu lösen und somit den Wert h zu bestimmen, der für die Berechnung der Anzahl Auswertungen in der zweiten Stufe benötigt wird.

6.3.3.3. ISS

Die Klasse *ISS* implementiert die *Iterative Subset Selection*, wie sie in Abschnitt 6.2.5 vorgestellt wurde. Dem Konstruktor werden hier die *BlackBox*, der *indifferenceAmount* sowie die minimale Anzahl initialer Auswertungen und die approximierte Signifikanz übergeben.

Das eigentliche Auswahlverfahren wurde in der Methode *determineBestConfigurations* implementiert. Um die *Extended screen-to-the best*-Prozedur durchzuführen wurde die private Methode *ESTTB* hinzugefügt, die von der Hauptmethode aus aufgerufen wird.

6.3.3.4. ISS_CRN

Die Klasse *ISS_CRN* verwirklicht die verbesserte Version der *Iterative Subset Selection*, die für Common Random Numbers angepaßt wurde. Abgesehen von der *ESTTB*-Methode, in der die Anpassung tatsächlich stattgefunden hat, funktioniert hier alles genause wie bei *ISS*.

6.3.3.5. KoenigAndLaw

Das Verfahren nach Koenig und Law wurde in der Klasse *KoenigAndLaw* (siehe 6.2.6) implementiert. Auch hier werden dem Konstruktor ein Zeiger auf die *BlackBox*, der *indifferenceAmount*, die Anzahl Auswertungen für die erste Stufe sowie die Schranke für die Wahrscheinlichkeit einer korrekten Auswahl übergeben.

Die grundlegende Funktionalität wurde in der Methode *determineBestConfigurations* implementiert, die für die Berechnung der Anzahl Auswertungen der zweiten Stufe diverse private Unterprozeduren aufruft. Im Gegensatz zu den restlichen Verfahren werden die Indizes der Konfigurationen nach erfolgten Auswertungen nicht nach der normalen Mittelwertschätzung sondern nach einer gewichteten Mittelwertschätzung über die beiden Stufen sortiert.

6.3.3.6. Means

Die Klasse *Means* realisiert die einfachste Strategie, nämlich die Auswahl nach einer festen Anzahl Auswertungen, wie sie in Abschnitt 6.2.2 beschrieben wurde. Ihrem Konstruktor wird neben der *BlackBox* ausschließlich die Anzahl Auswertungen, die für jede Konfiguration durchgeführt werden sollen, übergeben.

Auch hier wurde in der Methode *determineBestConfigurations* der eigentliche Algorithmus implementiert.

6.3.3.7. SSM

Die *Subset Selection with Memory* (siehe Abschnitt 6.2.7) wurde in der Klasse *SSM* verwirklicht. Dem Konstruktor müssen die *BlackBox*, der *indifferenceAmount*, die Anzahl initialer Auswertungen sowie die Schranke für die Wahrscheinlichkeit einer korrekten Auswahl übergeben werden.

Das eigentlich Ranking & Selection findet in der Methode *determineBestConfigurations* statt, wo der gesamte Algorithmus implementiert wurde.

6.4. Leistungsstudien

6.4.1. Einleitung

Christian Horoba

In diesem Abschnitt werden die Ergebnisse der Untersuchungen der verschiedenen Auswahlverfahren in Verbindung mit einer Evolutionsstrategie bei der Optimierung unterschiedlicher Funktionen dargestellt. Es wurden sowohl künstliche Testfunktionen als auch einfache Simulationsmodelle betrachtet.

Eingangs werden die drei betrachteten Testfunktionen und die beiden untersuchten Simulationsmodelle beschrieben. Anschließend wird die zur Optimierung verwendete Evolutionsstrategie vorgestellt. Danach folgt die Erörterung der Untersuchungsergebnisse.

6.4.2. Testfunktionen

Christian Horoba

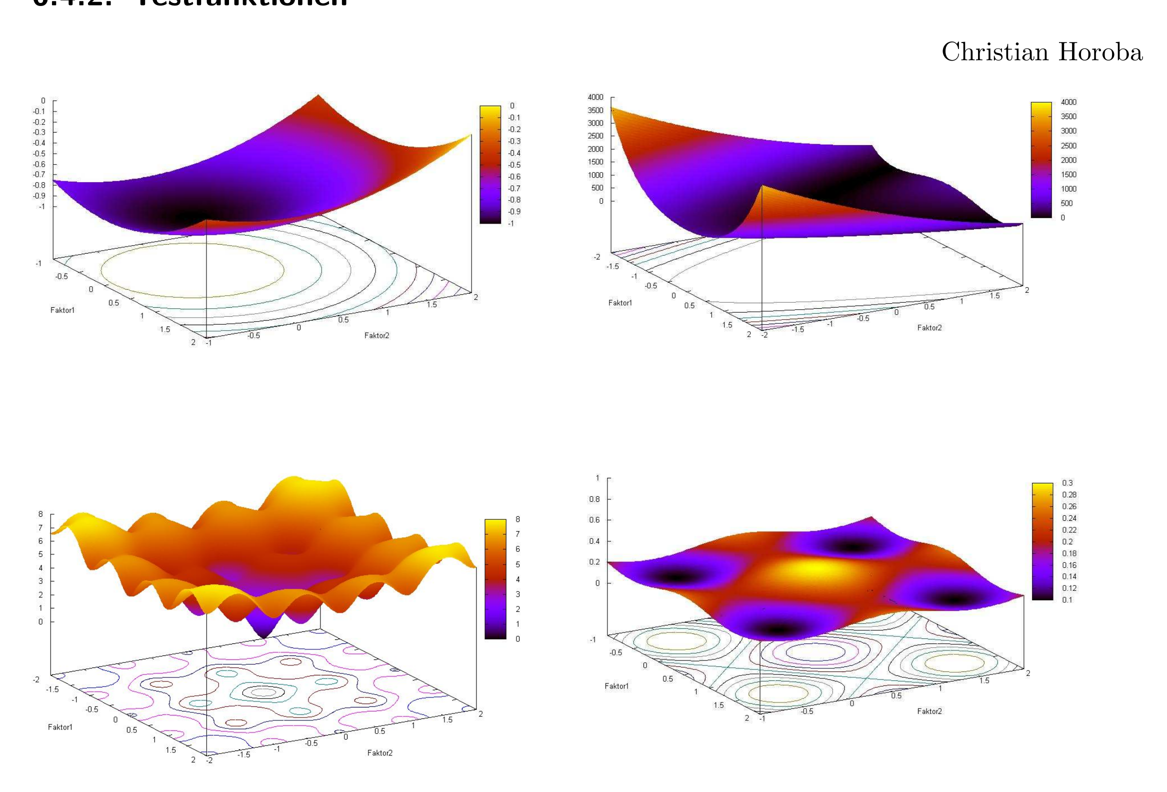


Abb. 6.5.: Oberflächen: Sphere-Funktion (l.o.), Rosenbrocks Funktion (r.o.), Ackleys Funktion (l.u.), Rauschstärke-Funktion (r.u.).

Die folgenden drei bekannten Testfunktionen kamen zum Einsatz:

$$f_{\text{Sphere-Funkt.}} : [-1,2]^2 \to [0,1]\,, \mathbf{x} \mapsto \frac{1}{8}\left(x_1^2 + x_2^2\right) - 1,$$

$$f_{\text{Rosenbrocks Funkt.}} : [-2,2]^2 \to [0,3609]\,,\, \mathbf{x} \mapsto 100\left(x_1^2 + x_2\right)^2 + (1-x_1)^2\,,$$

$$f_{\text{Ackleys Funkt.}} : [-2,2]^2 \to [0,8]\,,\, \mathbf{x} \mapsto -20e^{-\frac{1}{5}\left(\frac{1}{2}\left(x_1^2+x_2^2\right)\right)^{\frac{1}{2}}} - e^{\frac{1}{2}(\cos(2\pi x_1)+\cos(2\pi x_2))} + 20 + e.$$

Man beachte, daß alle Testfunktionen ihr globales Minimum in dem Punkt $(0,0)$ annehmen, wobei der Funktionswert dann 0 beträgt. Die Testfunktionen werden in Abbildung 6.5 veranschaulicht.

Um dem stochastischen Rauschen typischer Simulationsmodelle Rechnung zu tragen, addieren wir normalverteilte Zufallsgrößen auf die Funktionswerte der Testfunktionen. Die Stärke des Rauschens wird durch die Rauschstärke-Funktion g festgelegt. Insgesamt erhalten wir folgende Testfunktion:

$$f_{sim}(x) \mapsto f(x) + N(0,1)\,g(x)\,,$$

wobei $N(0,1)$ eine standardnormalverteilte Zufallsgröße bezeichnet. In allen Experimenten kam folgende Rauschstärke-Funktion zum Einsatz:

$$g : (x_1,\ldots,x_n) \mapsto \left(1 + \frac{1}{2n}\sum_{i=1}^{n}\sin(\gamma\pi x_i)\right)\sigma,$$

wobei σ die Rauschstärke und γ die Frequenz der Rauschstärkenschwankung bezeichnet. Man beachte, daß g Funktionswerte von 0.5σ bis 1.5σ annimmt. In den nachfolgenden Leistungsstudien wurde der Parameter σ – sofern nicht anders angegeben – auf 10% der Größe des Wertebereichs der Testfunktion gesetzt. Der Parameter γ wurde für alle Leistungsstudien auf den Wert 1 gesetzt. Die Rauschstärke-Funktion wird in Abbildung 6.5 (r.u.) dargestellt.

6.4.3. Simulationsmodelle

Christian Horoba

Es wurden zwei typische Simulationsmodelle untersucht: die *Fertigungsstraße (Tandem-Warteschlange)* und das *(s,S)-Lagerhaltungssystem*.

6.4.3.1. Fertigungsstraße

Bei diesem Modell handelt es sich um N in Reihe geschaltete Wartesysteme.

Objekte, die ein Wartesystem verlassen, werden ohne Zeitverzögerung dem nächsten Wartesystem zugeführt. Die Zwischenankunftszeiten neuer Objekte am ersten Wartesystem sind exponentialverteilt mit Rate 0.5. Die Bedienzeiten der einzelnen Bediener sind ebenfalls exponentialverteilt mit Rate μ_i. Diese Warteschlangen verfügen über eine Kapazität von 10 Objekten. Objektankünfte bei vollen Warteschlangen werden ignoriert. Die Objekte werden gemäß der Bediendisziplin *first-come, first-serve* (FCFS) bedient. Der Optimierer steht vor der Aufgabe, geeignete Bedienraten $\mu = (\mu_1,\ldots,\mu_N)$ zu bestimmen, so daß der größtmögliche Gewinn erzielt wird. Der Gewinn wird durch folgende Gewinnfunktion bestimmt:

$$R(\mu) = \frac{r \cdot X(\mu)}{c_0 + \mathbf{c}^T \cdot \mu} - c_1,$$

wobei $X(\mu)$ den Durchsatz des Systems mit den Bedienraten μ bezeichnet und r für einen Gewinnfaktor, c_0, c_1 für Basiskosten und $\mathbf{c}$ für Kostenfaktoren für die einzelnen Bediensysteme stehen.

In den durchgeführten Experimenten wurden $N = 3$ Wartesysteme betrachtet. Anfangs befanden sich keine Objekte im System. Die transiente Simulation endete nach 1000 Zeiteinheiten. Des Weiteren wurden folgende Parametereinstellungen fest gewählt: $r = 10000$, $c_0 = 1$, $c_1 = 400$ und $\mathbf{c}^T = (1, 5, 9)$. Der Suchraum wurde auf die Menge $[0.1, 2]^3$ eingeschränkt.

Die möglichen Gewinne lagen dann ungefähr zwischen -400 und 98.5.

6.4.3.2. (s,S)-Lagerhaltungssystem

Dieses Modell formalisiert ein Lagerhaltungssystem und ist [14] entnommen.

Die beiden zu optimierenden Größen werden mit s und S bezeichnet. Falls zu Beginn einer Periode der Lagerbestand b kleiner als s ist, wird eine Bestellung der Größe $S - b$ getätigt, die nach u Perioden eintrifft, wobei u uniform zufällig aus dem Intervall $[0.5, 1]$ gezogen wird. Die Zeit zwischen zwei Kundenbestellungen ist exponentialverteilt mit Rate 0.1. Die Kunden fragen mit Wahrscheinlichkeit $\frac{1}{6}$ eine Ware, mit Wahrscheinlichkeit $\frac{1}{3}$ zwei Waren, mit Wahrscheinlichkeit $\frac{1}{3}$ drei Waren und mit Wahrscheinlichkeit $\frac{1}{6}$ vier Waren nach. Kundenbestellungen werden stets erfüllt. Falls sich dadurch negative Lagerbestände einstellen, fallen zusätzliche Kosten an (s.u.). Die Aufgabe des Optimierers ist – durch geeignete Wahl der Parameter s und S – die durchschnittlichen Kosten C pro Periode zu minimieren, die sich folgendermaßen zusammensetzen:

$$C = c_B + c_L + c_Z,$$

wobei c_B die durchschnittlichen Bestellkosten pro Periode, c_L die durchschnittlichen Lagerkosten pro Periode und c_Z die durchschnittlichen Zusatzkosten im Fall negativer Lagerbestände bezeichnen. Die genannten Einzelkosten ergeben sich folgendermaßen, wobei die hier verwendeten Kostenfaktoren schon eingesetzt sind:

$$c_B = \text{durchschnittl. Bestellanzahl/Periode} \cdot 32 + \text{durchschnittl. Bestellmenge/Periode} \cdot 3,$$

$$c_L = \text{durchschnittlicher positiver Lagerbestand/Periode} \cdot 1,$$

$$c_Z = \text{durchschnittlicher negativer Lagerbestand/Periode} \cdot 5.$$

Das System verfügte zu Beginn der Simulation über einen Lagerbestand von 60 Einheiten. Die transiente Simulation umfaßte 120 Perioden. Als Suchraum wurde die Menge $[0.1, 120]^2$ gewählt. Abbildung 6.6 stellt die Oberfläche der Kostenfunktion über dem betrachteten Definitionsbereich dar.

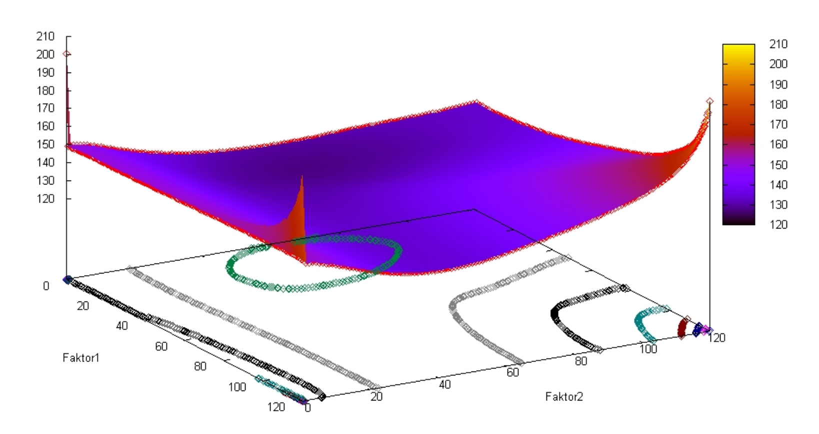


Abb. 6.6.: Oberfläche der Kostenfunktion des *(s,S)-Lagerhaltungssystems* (1000 Replikationen).

6.4.4. Verwendete Evolutionsstrategien

Christian Horoba

Dieser Abschnitt beschreibt die verwendeten Evolutionsstrategien. Es wurden allgemein eine $(5+5)$- und in einzelnen Untersuchungen eine $(5,10)$-Evolutionsstrategie benutzt. Die Größe der Elitepopulation wurde auf 10 gesetzt. Die Eltern wurden uniform zufällig gewählt, wobei die mehrfache Auswahl eines Individuums erlaubt wurde. Es kam kein Rekombinationsoperator zum Einsatz. Die Mutation erfolgte durch Addition von normalverteilten Zufallsgrößen, wobei die Mutationsstärken automatisch mittels Selbstadaption angepaßt wurden. Die Umweltselektion verlief deterministisch. Der Algorithmus wurde nach Berechnung von 100 Generationen (im Fall der Simulationmodelle 150 Generationen) abgebrochen. Teilweise wurde auch nach 10000 durchgeführten Funktionsauswertungen die Optimierung beendet.

Die statistischen Auswahlverfahren kamen nur bei der Umweltselektion zum Einsatz. Für die Verwaltung der Elitepopulation und die abschließende Auswahl eines besten Individuums aus der Elitepopulation wurden keine zusätzlichen Auswertungen angestoßen.

Die Auswahlverfahren wurden standardmäßig folgendermaßen eingestellt: Die Anzahl (initialer) Auswertungen n_0 wurde auf 10 gesetzt. Der Gleichheitsbereich-Parameter δ wurde auf 10% der Größe des Wertebereichs der betrachteten Testfunktion gesetzt. Im Falle der Simulationsmodelle wurde dieser geschätzt. Die verwendeten Werte lauten konkret:

- $\delta_{\text{Sphere-Funktion}} = 0.1$,
- $\delta_{\text{Rosenbrocks Funktion}} = 360.9$,
- $\delta_{\text{Ackleys Funktion}} = 0.8$,
- $\delta_{\text{Tandem-Warteschlange}} = 12.5$,
- $\delta_{\text{(s,S)-Lagerhaltungssystem}} = 2.5$.

Die Wahrscheinlichkeitsschranke P wurde auf 0.5 gesetzt. Die beiden letztgenannten Parameter wurden sehr moderat eingestellt, um exzessive Auswertungsanzahlen zu vermeiden.

6.4.5. Untersuchung der Testfunktionen

Christian Horoba

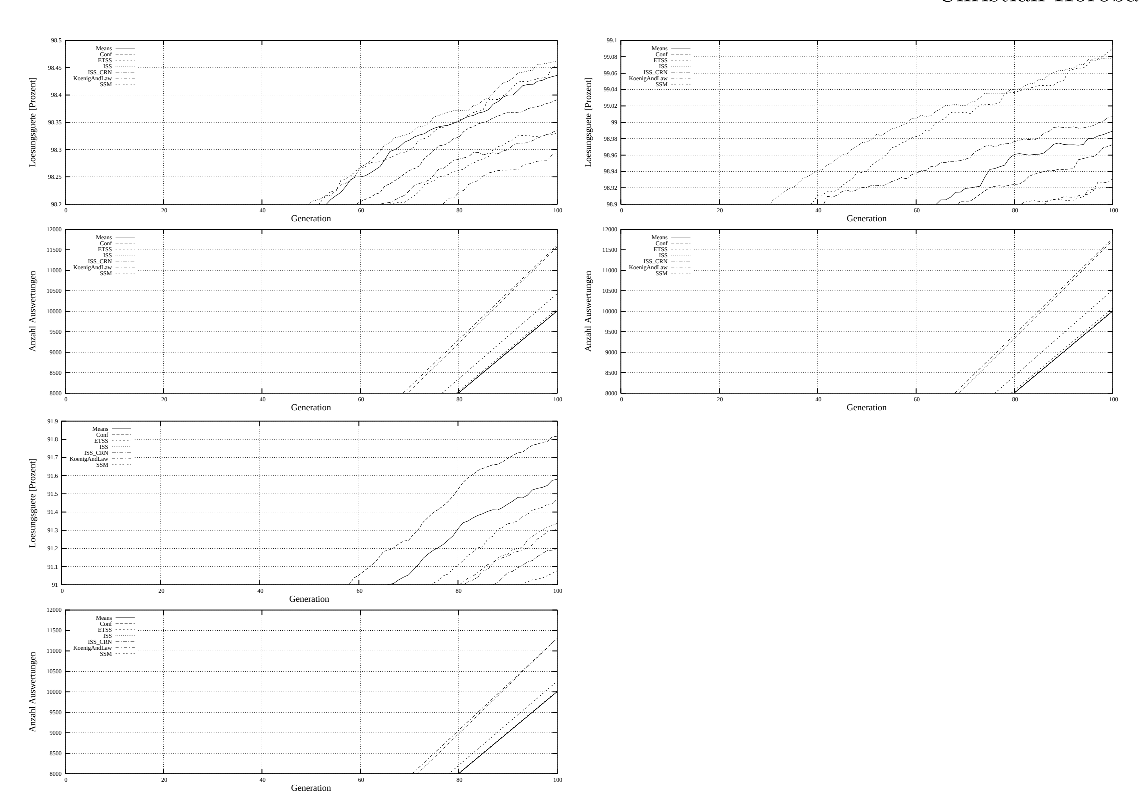


Abb. 6.7.: Lösungsgüte und Anzahl Auswertungen aller Auswahlverfahren im Vergleich: Sphere-Funktion (l.o.), Rosenbrocks Funktion (r.o.), Ackleys Funktion (l.u.) (2500 Replikationen).

Die in Abbildung 6.7 dargestellten Diagramme geben eine erste Übersicht über den Nutzen der betrachteten statistischen Auswahlverfahren. In den folgenden Ausführungen bezeichnet Means das Mittelwert-Auswahlverfahren und Conf das Konfidenzintervall-Auswahlverfahren.

Bei der Sphere-Funktion resultiert die Verwendung der Auswahlverfahren ISS, SSM und Means in den besten Lösungen (siehe Abbildung 6.7 (l.o.)). Hierbei muß berücksichtigt werden, daß das Verfahren ISS deutlich mehr Auswertungen als die anderen beiden genannten Verfahren durchführt (ISS: 11568.9, SSM: 10428.6, Means: 10005).

Bei Rosenbrocks Funktion erhält man durch die Verwendung der Verfahren ISS, ETSS und ISS (für CRNs) die besten Ergebnisse (siehe Abbildung 6.7 (r.o.)). Auch hier muß

bezüglich der Auswertungsanzahlen ein ähnliches Gefälle wie oben berücksichtigt werden (ISS: 11736.5, ETSS: 10082, ISS (für CRNs): 10017.7).

Bei Ackleys Funktion bewähren sich die Verfahren Conf, Means und SSM (siehe Abbildung 6.7 (l.u.)). Die Anzahlen der durchgeführten Auswertungen unterscheiden sich nur wenig (Conf: 10005.2, Means: 10005, SSM: 10259.4).

Die Diagramme unterstützen die Vermutung, daß das Auswahlverfahren nach Koenig und Law in dem betrachteten Kontext mit den getesteten Einstellungen nicht empfehlenswert ist, da trotz der hohen Auswertungsanzahlen nur vergleichsweise schlechte bis mittelmäßige Lösungen gefunden werden. Des Weiteren ist erkennbar, daß das Verfahren ISS weniger gute Resultate bei der multimodalen Ackley-Funktion erzielt, da hier trotz der höchsten Auswertungsanzahl von 11314 nur eine relativ mittelmäßige Lösung gefunden wird.

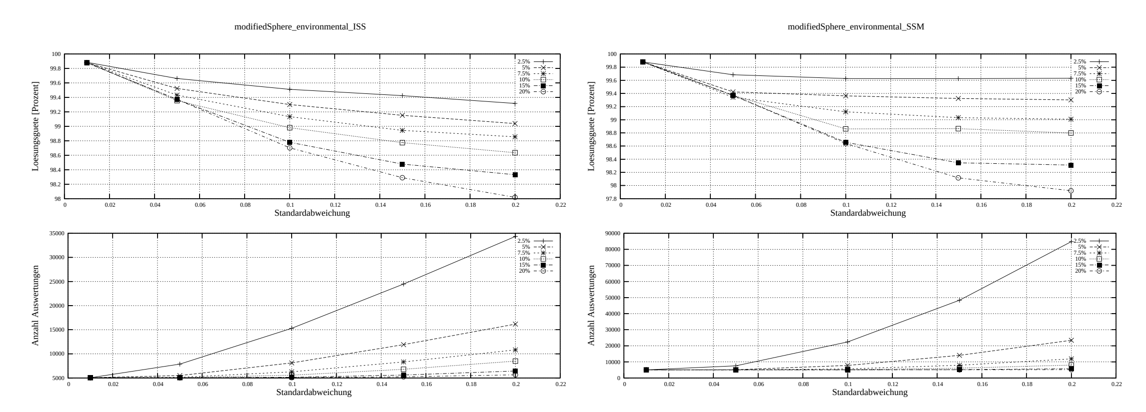


Abb. 6.8.: Auswirkungen des Gleichheitsbereich-Parameters auf die Verfahren ISS (l.) und SSM (r.) bei verschiedenen Rauschstärken und der Optimierung der Sphere-Funktion (2500 Replikationen).

Die nächste Studie untersucht die Auswirkungen der Wahl des Gleichheitsbereich-Parameters auf die Auswahlverfahren bei verschiedenen Rauschstärken. Abbildung 6.8 zeigt die Resultate für die Verfahren ISS und SSM und die Optimierung der Sphere-Funktion. Die Meßwerte befinden sich bei Rauschstärken, die bezogen auf den Wertebereich der Funktion 1%, 5%, 10%, 15% und 20% entsprechen. Die Größe der Gleichheitsbereich-Parameter wird ebenfalls prozentual auf die Größe des Wertebereichs bezogen.

Allgemein kann man feststellen, daß bei zunehmender Stärke des Rauschens die Lösungsgüten abnehmen und die Auswertungsanzahlen zunehmen. Außerdem gilt allgemein, daß ein kleinerer Gleichheitsbereich-Parameter zu besseren Lösungsgüten und mehr Auswertungen führt. Des Weiteren scheint sich die Wirkung der Parameterwahl bei stärker verrauschten Funktionen tendenziell zu verstärken.

Man kann die Empfehlung ableiten, den Parameter moderat zu wählen, um exzessive Auswertungen zu vermeiden.

In der folgenden Studie wurden die Beeinflussungen der Ergebnisse durch unterschiedliche Größen der Elitepopulation untersucht. Es wurden die Elitepopulationsgrößen 1, 5, 10, 25 und 50 im Zusammenhang mit allen Auswahlverfahren und allen Testfunktio-

nen untersucht. Des Weiteren wurde in diese Untersuchungen neben der $(5 + 5)$- eine $(5, 10)$-Evolutionsstrategie einbezogen.

Die Ergebnisse zeigen, daß die Größe der Elitepopulation bei den verwendeten Konfigurationen keine Auswirkungen auf die Lösungsgüten und Auswertungsanzahlen zu haben scheint. Außerdem wurde festgestellt, daß die Verwendung der Plus-Umweltselektion zu besseren Ergebnissen als die Verwendung der Komma-Umweltselektion führt.

6.4.6. Terminierung nach 10000 Auswertungen

Peter Kissmann

Zur Untersuchung des Verhaltens der Evolutions-Strategie auf den drei Testfunktionen mit 10000 Auswertungen als Abbruchkriterium wurden hier wieder die sieben verschiedenen Ranking & Selection-Verfahren miteinander verglichen. Desweiteren wurde das Rauschen der Funktionen variiert, um Aussagen darüber treffen zu können, wie sich die Verfahren bei unterschiedlich starkem Rauschen verhalten. Die mittlere Standardabweichung des Rauschens wurde auf 0.1, 5, 10, 15 und 20 Prozent der Größe des Wertebereichs der jeweiligen Funktion eingestellt.

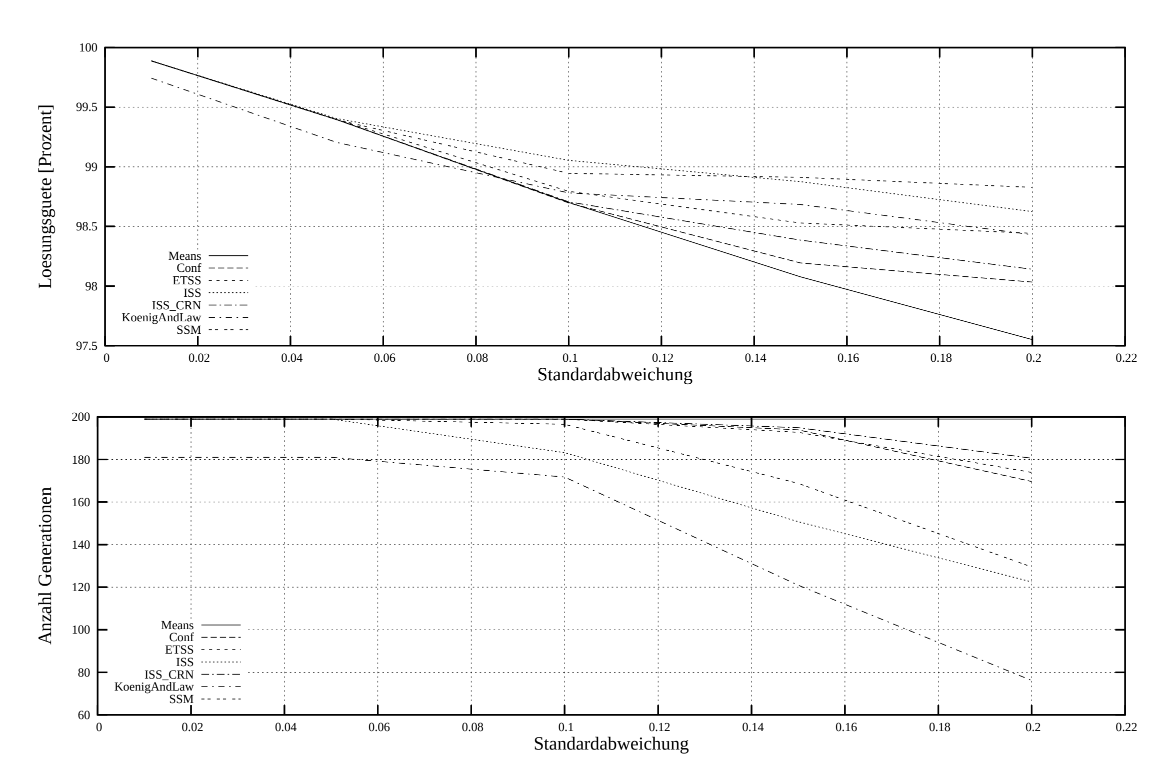


Abb. 6.9.: R&S-Verfahren auf der Sphere mit Abbruch nach 10000 Auswertungen (Plus-Strategie)

In Abbildung 6.9 wird das Verhalten auf der Sphere-Funktion mit Plus-Umweltselek-

tion dargestellt. Es läßt sich gut erkennen, daß bei geringen Rauschstärken das Verfahren nach Koenig und Law ein wenig schlechter als die restlichen ist. Das liegt daran, daß dieses Verfahren in der zweiten Stufe stets mindestens eine Auswertung macht, die anderen Verfahren jedoch bei sehr geringer Rauschstärke nicht. Somit ist es nicht mehr in der Lage, so viele Generationen zu durchlaufen wie die restlichen und erzielt im Endeffekt eine schlechtere Lösungsgüte.

Sobald das Rauschen jedoch relativ stark wird, fällt auf, daß die Verfahren, die die meisten Generationen durchlaufen haben und somit die einzelnen Individuen am seltensten ausgewertet haben, schlechter sind als jene, die weniger Durchläufe absolviert haben. So werden bei *SSM* nur etwa 120 Generationen berechnet und eine Fitneß von etwa 98, 8 Prozent erzielt, wohingegen *Means* 200 Generationen durchläuft, aber nur eine Fitneß von etwa 97, 5 Prozent erreicht. Man kann also sagen, daß es bei starkem Rauschen und Plus-Umweltselektion besser ist, ein Individuum häufiger auszuwerten als viele Generationen zu durchlaufen und dabei zwar mehr verschiedene Individuen zu betrachten aber diese seltener auszuwerten. Vermutlich wird im letztgenannten Fall ein Individuum nicht häufig genug ausgewertet, um das Rauschen hinreichend zu unterdrücken, so daß recht häufig Individuen zum Überleben ausgewählt werden, die eigentlich schlechter als andere sind.

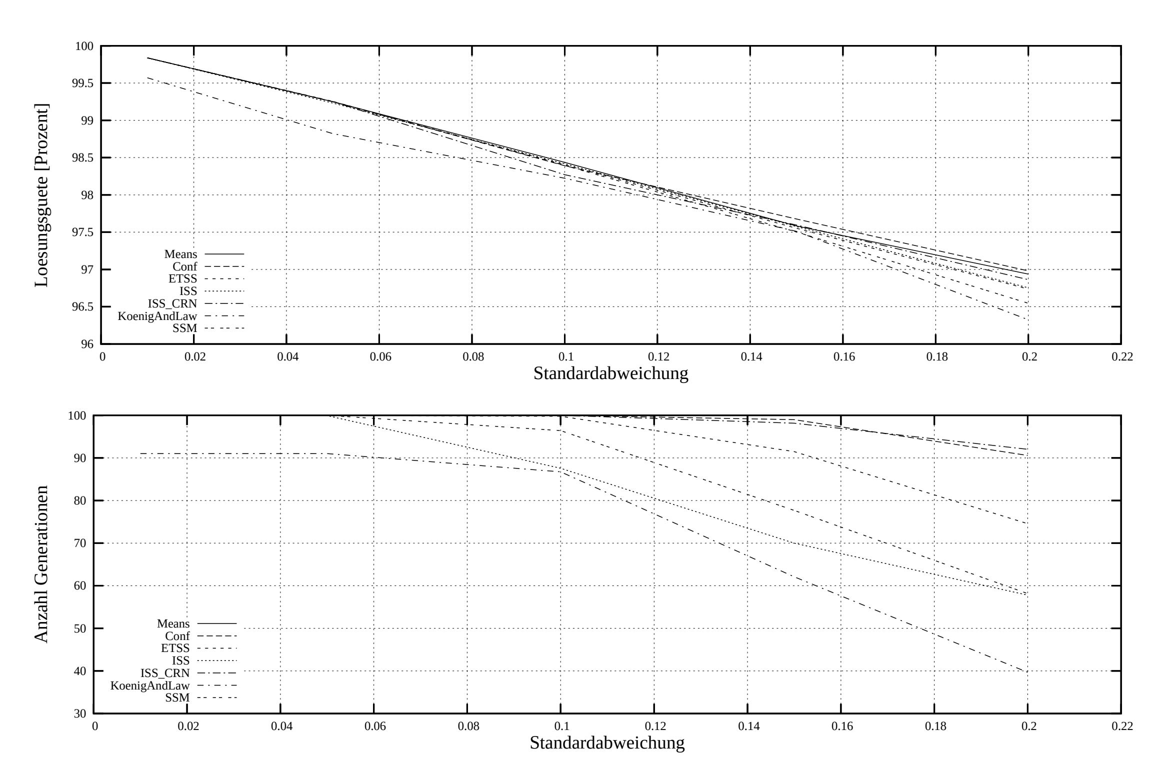


Abb. 6.10.: R&S-Verfahren auf der Sphere mit Abbruch nach 10000 Auswertungen (Komma-Strategie)

Bei der Komma-Umweltselektion, die in Abbildung 6.10 dargestellt ist, ist das Verhalten bei schwachem Rauschen sehr ähnlich: Auch hier sind alle Verfahren gleich gut und durchlaufen die gleiche Menge an Generationen, abgesehen vom Verfahren nach Koenig und Law. Die Begründung dafür ist hier natürlich die selbe wie zuvor.

Ein großer Unterschied findet sich jedoch bei hohen Rauschstärken. Hier sind die beiden einfachen Heuristiken *Means* und*Conf* am Besten, obwohl gerade *Means* die meisten Generationen schafft und somit die Individuen am wenigsten auswertet. Bei der Komma-Selektion scheint es also eher so zu sein, daß viele Generationen entscheidender für eine gute Fitneß sind als viele Auswertungen pro Individuum.

Diese Beobachtungen bestätigen sich auch auf den beiden anderen getesteten Funktionen, könnten also durchaus allgemein gelten.

6.4.7. Leistungsstudien auf Simulationsmodellen

Peter Kissmann

Für den Vergleich der Ranking & Selection-Verfahren auf Simulationsmodellen wurden die beiden als C++-Programme vorliegenden Modelle des Lagerhaltungssystems und der Fertigungsstraße (Tandem-Warteschlange) gewählt, da hier die Möglichkeit besteht, Common Random Numbers zu aktivieren. Untersucht wurden hier die folgenden Fragen:

1. Wie verhalten sich die Verfahren auf den Simulationsmodellen mit beziehungsweise ohne Common Random Numbers?

2. Wie verhalten sich die Verfahren auf den Simulationsmodellen mit unterschiedlichen Gleichheitsbereich-Parameter-Einstellungen?

Diesen Fragen wird in den folgenden Unterabschnitten nachgegangen.

6.4.7.1. Untersuchung von Common Random Numbers

In Abbildung 6.11 ist die Untersuchung des Verhaltens von Ranking & Selection-Verfahren auf der Fertigungsstraße mit beziehungsweise ohne Common Random Numbers dargestellt. Im Falle, daß die Common Random Numbers deaktiviert sind, benötigen die meisten Verfahren etwa 8000 bis 10000 Auswertungen; das Verfahren nach Koenig und Law benötigt etwa 12000 Auswertungen. Die Lösungsgüte liegt bei den Verfahren in einem Bereich von etwa 87 bis 90. Die numerisch bestimmte beste Lösung hat eine Güte von etwa $98,46$, so daß man sagen kann, daß die Verfahren noch relativ weit vom Optimum entfernt sind.

Demgegenüber läßt sich im Falle aktivierter Common Random Numbers erkennen, daß bei den meisten Verfahren die Anzahl Auswertungen auf das Niveau von *Means* gefallen ist (wenn sie nicht schon vorher dort lag). Die einzigen Ausnahmen bilden zum einen das Verfahren nach Koenig und Law, das weiterhin etwa 12000 Auswertungen benötigt hat, sowie die ursprüngliche *ISS*-Prozedur, die nun statt der ehemals etwa 9000 Auswertungen über 25000 Stück durchführt. Dies ist damit zu begründen, daß diese Prozedur nicht für Common Random Numbers geeignet ist. Daher wurde sie derart angepaßt,

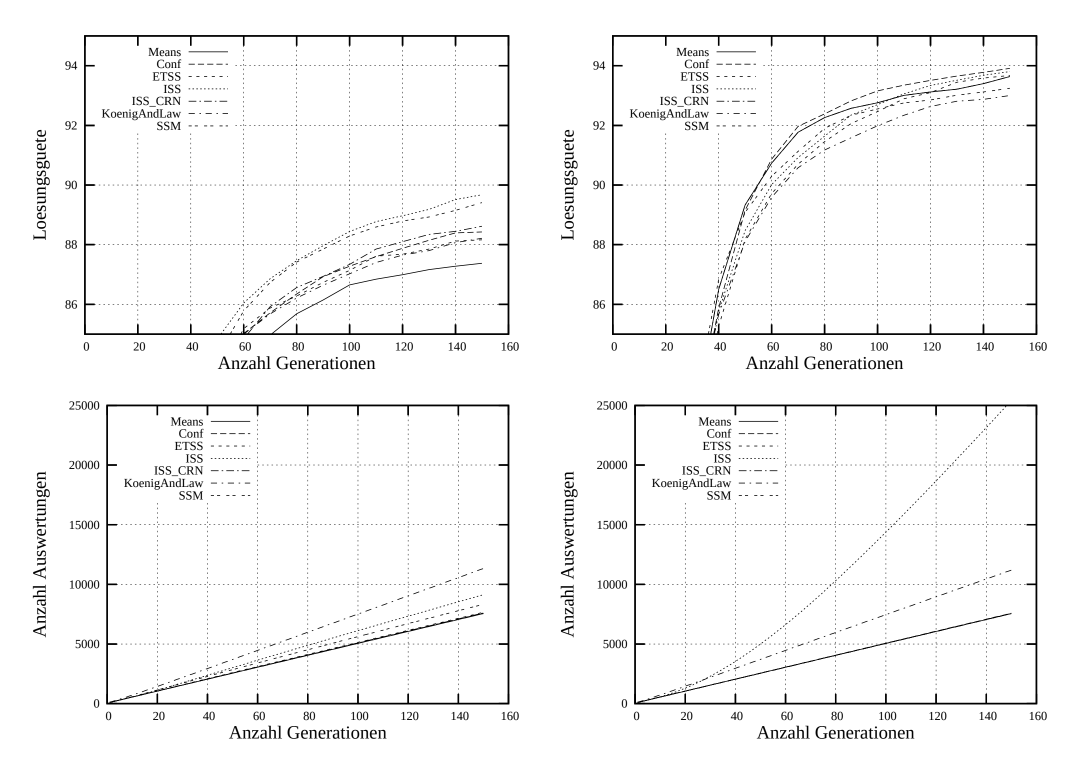


Abb. 6.11.: R&S-Verfahren auf der Fertigungsstraße links ohne, rechts mit Common Random Numbers

daß sie auch mit Common Random Numbers gut zurecht kommt, was auf das Verfahren *ISS_CRN* geführt hat (siehe Abschnitt 6.2.5). Dieses Verfahren funktioniert damit tatsächlich deutlich besser; es benötigt ebenfalls nur noch die etwa 8000 Auswertungen, die auch die meisten anderen Verfahren durchführen. Die Lösungsgüte ist trotz der teils geringeren Anzahl an Auswertungen erheblich besser geworden: Alle Verfahren erzielen eine Lösung im Bereich von etwa 93 bis 94, was deutlich näher am Optimum liegt.

Ein sehr ähnliches Verhalten hat sich auch bei dem (s, S)-Lagerhaltungssystem gezeigt: Auch hier ändert sich beim Verfahren nach Koenig und Law nichts an den Auswertungsanzahlen. Ebenfalls ließ sich hier der ungeheure Zuwachs an Auswertungen im Falle eingeschalteter Common Random Numbers bei der *ISS*-Prozedur beobachten. Und ebenso wie bei dem Fertigungssystem ist auch hier *ISS_CRN* sowohl mit als auch ohne Common Random Numbers gut zurecht gekommen. Was hier allerdings auffiel, ist, daß sich die Lösungsgüte zwischen aktivierten und deaktivierten Common Random Numbers nur geringfügig unterschied. Alles in allem haben die Verfahren nur eine geringe Lösungsgütenverbesserung aufgrund der Common Random Numbers erfahren.

Es läßt sich festhalten, daß Common Random Numbers bei einigen Modellen sehr gut funktionieren und die dafür optimierten Verfahren deutlich weniger Auswertungen benötigen, um eine bessere Lösungsgüte zu erzielen.

6.4.7.2. Untersuchung des Gleichheits-Bereich Parameters

Hier wurden verschiedene Werte für den Gleichheits-Bereich Parameter gesetzt, nämlich jeweils etwa 2.5, 5, 7.5, 10, 15 und 20 Prozent der Größe des Wertebereiches. Für die Fertigungsstraße wurde hierbei angenommen, daß der Wertebereich etwa von −400 bis +100 reicht, also eine Größe von 500 hat; für das Lagerhaltungssystem wurde eine Größe von etwa 100 angenommen. Was bei all diesen Untersuchungen (sowohl mit als auch ohne Common Random Numbers) auffiel, ist in Abbildung 6.12 dargestellt: Bei geringem

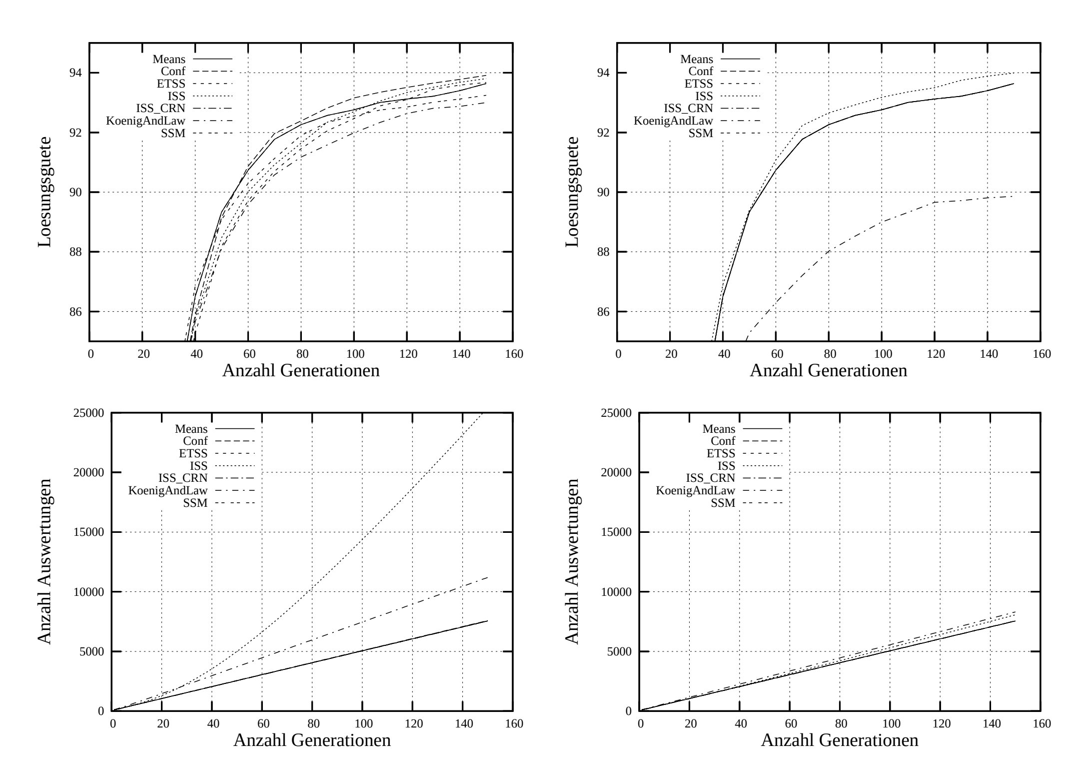


Abb. 6.12.: R&S-Verfahren auf der Fertigungsstraße links Gleichheits-Bereich Parameter etwa 2, 5 Prozent des Wertebereiches, rechts etwa 7, 5 Prozent (jeweils mit Common Random Numbers)

Gleichheits-Bereich Parameter sind die Lösungsgüten der Verfahren relativ unterschiedlich (bei etwa 2, 5 Prozent und der Fertigungsstraße im Bereich von etwa 93 bis 94). Sobald er allerdings hoch gesetzt wird, stimmen fast alle Kurven exakt mit der von *Means* überein. Bereits bei einer moderaten Einstellung von etwa 7, 5 Prozent ist dies der Fall für alle Verfahren außer *KoenigAndLaw* und (bei eingeschalteten Common Random Numbers) *ISS*. Bei noch höheren Einstellungen ist auch *ISS* absolut identisch mit den restlichen Verfahren (außer *KoenigAndLaw*), so daß hier selbst das schlechte Abschneiden bei eingeschalteten Common Random Numbers nicht mehr ins Gewicht fällt. Dies läßt sich dadurch erklären, daß die Verfahren in keiner einzigen Replikation mehr

Auswertungen der einzelnen Individuen als die initiale Anzahl durchgeführt haben, was darauf zurückzuführen ist, daß die untersuchten Simulationsmodelle sehr schwach verrauscht sind. Damit sind die Zufallsströme stets vollkommen identisch und die Werte stimmen exakt überein. Einzig das Verfahren nach Koenig und Law, das in der zweiten Stufe stets mindestens eine Auswertung durchführt, stimmt somit nicht mit den anderen überein. Interessanterweise ist es hierbei so, daß die eine zusätzliche Auswertung keinen Vorteil bringt, sondern dieses Verfahren sogar deutlich schlechtere Lösungen liefert als die restlichen (eine Lösungsgüte von etwa 90 gegenüber einer von fast 94). Dies mag darin begründet sein, daß dieses Verfahren einen gewichteten Mittelwert bildet, der die Mittelwertschätzung stark verzerrt.

7. Kriging Metamodelle

Igor Gudovsikov, Michael Schaten

7.1. Einführung in Kriging

Viele Simulations-Experimente beanspruchen eine lange Berechnungszeit während der Ausführung. Man versucht daher, die Anzahl von Simulations-Auswertungen so gering wie möglich zu halten. Dennoch soll natürlich die Suche nach einem Optimum (im folgenden wird stets von einer Minimierung ausgegangen) erfolgreich verlaufen, sprich: Das Finden des Optimums soll auch dann noch möglich sein, wenn nicht alle möglichen Faktorkombinationen berechnet werden (wie es beispielsweise bei einem Full factorial Design bei der Response Surface Methode [RSM siehe Kapitel 3, RSM-Design siehe Kapitel 3.1.3] praktiziert wird). Eine Lösung aus diesem Dilemma stellt Interpolation, die Basisidee von Kriging, dar.

Der Name Kriging geht zurück auf den südafrikanischen Bergbau-Ingenieur D. G. Krige. Es wurde Mitte des letzten Jahrhunderts von G. Matheron in Frankreich zur Anwendung im Bereich Bergbau zur Vorhersage von Gold-Vorkommen entwickelt. Zur gleichen Zeit entwickelte L.S. Gandin in der Sowjetunion das selbe Verfahren und wandte es auf den Bereich der Meteorolgie an.

Betrachtet man andere Verfahren zur Approximation von Response Surfaces, wie beispielsweise die Approximation durch lineare Regression, im Vergleich zu Kriging Metamodellen, so ist die Idee der Interpolation offensichtlich.

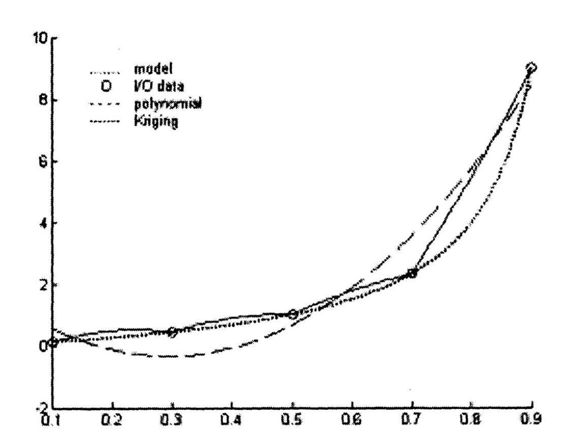


Abb. 7.1.: Kriging im Vergleich zur polynomiellen Regression zweiter Ordnung in einem Simulations-Experiment

In Abbildung 7.1 ist der Response zu einer gegebenen Variablenbelegung aufgetragen. Insgesamt gibt es fünf tatsächliche Auswertungen (symbolisiert durch Kringel) durch einen Simulator.

Die Kennlinien der beiden Approximationsideen drücken deren jeweilige Idee klar aus: Die Approximation durch eine polynomielle Ordnung zweiter Ordnung (lang gestrichelte Linie) hat offensichtlich zum Ziel, den erwarteten Fehler „im Ganzen“ (global) minimal zu halten. Hingegen erkennt man beim Kriging Metamodell (durchgezogene Linie), dass hier die Design-Punkte absolut mit den im Metamodell vorkommenden Punkten übereinstimmen. Zwischen den Design-Punkten wird dann interpoliert und dies - zumindest zwischen den ersten vier Punkten - sehr gut, denn der Vergleich zur originalen Kurve (gepunktete Linie) zeigt einen hohen Grad an Übereinstimmung.

Nun ist natürlich sorgfältig darüber zu überlegen, mittels welchen Designs man die zur Interpolation nötigen Design-Punkte bestimmt. Es existieren einige für Kriging geeignete Designs, von denen sich jedoch in der Praxis bzw. der Projektgruppe lediglich LHS (=Latin Hypercube Sampling) durchsetzen konnte. Details zum Design mittels LHS werden in Kapitel 7.2.1 beschrieben.

7.2. Die Technik hinter Kriging

7.2.1. Designs für Kriging

Das vorhergehende Kapitel beschränkte sich auf die Kriging Analyse. Im Folgenden wird das Design des Experiments zur Analyse eines Simulations-Modells durch Kriging beschrieben.

Die einfachsten und auch erfolgreichsten Designs benutzen LHS (=Latin Hypercube Sampling), ein sogenanntes *space filling design*. LHS wurde 1979 von McKay, Beckman und Conover für deterministische Simulationsmodelle entwickelt.

Der größte Vorteil, den LHS bietet, ist die flexible Design-Größe n (Anzahl der tatsächlich simulierten Input-Kombinationen) für jedes k (Anzahl der zu untersuchenden Faktoren).

Damit auch randomisierte Simulationsmodelle (sinnvoll) approximiert werden können, muss auch bei eventuell teuren (=zeitaufwendigen) Simulations-Läufen, eine oder mehrere *Replikation(en)* möglich sein. Würde man für jeden Design-Punkt lediglich eine Auswertung als Basis für die Interpolation nutzen, hätte man (je nach Stärke des Rauschens) eventuell völlig nutzlose Response-Werte. Kleijnen und Van Beers [2004a] demonstrieren für Fälle, in denen zu wenige Replikationen im Design durchgeführt wurden, dass die Korrelations-Funktion (also auch die Kriging-Gewichte; Kap. 7.2.2) sehr ungenau wird. Folglich ist auch die „Vorhersage“ interpolierter Punkte durch den Predictor sehr ungenau. Für das folgende Beispiel (Abbildung 7.2) setzen wir also ausreichend viele Replikationen für die Design-Punkte voraus.

Zur Erstellung eines Designs per LHS, sind folgende Schritte durchzuführen:

1. Unterteilung aller k Eingabe-Wertebereiche in n gleich große, durchnummerierte Intervalle (die Wertebelegung für jeden Faktor kann also um ein Vielfaches größer sein, als bei z.B. klassischem full-factorial Design)

2. Platzieren der n Szenarienpunkte so, dass in jeder Spalte und jeder Zeile lediglich ein Punkt vorhanden ist.

3. In jeder der Zellen der Design-Matrix ist es nun noch möglich jeden der k Eingabewerte einzeln zu samplen (z.B. könnte es sinnvoll sein, einen der Werte an den Rand, also an einer extremen Stelle der Zelle, zu prüfen).

Betrachtet wird im folgenden Beispiel ein mögliches Design für zwei Faktoren (X_1, X_2) bei vier Szenarien. Tabelle 7.1 zeigt die zu der in Abbildung 7.2 gehörige Design-Matrix:

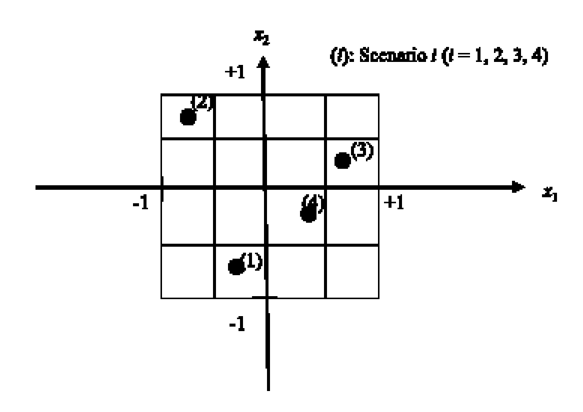


Abb. 7.2.: Ein LHS Design für zwei Faktoren und vier Szenarien

Sample	X_1	X_2
1	2	1
2	1	4
3	4	3
4	3	2

Tabelle 7.1.: Eine $LHS_{2,4}$ Design-Matrix für m=2 Variablen und n=4 Samples

Wie aus der LHS-Matrix ersichtlich, ist die Positionierung der Design-Punkte relativ einfach zu berechnen: Eine Permutation der Spalteneinträge gewährleistet die Eigenschaft, dass in jeder Spalte und Zeile des Designs nur genau ein Punkt vorhanden ist. Um bei diesem Beispiel zu bleiben:
Hier wird z.B. für das erste Szenario das zweite Intervall des Wertebereichs von X_1 mit dem ersten Intervall des Faktors X_2 kombiniert. Diese Kombination entspricht einer Permutation der n = 4 (n: Anzahl Intervalle in den Wertebereichen) von $2 \rightarrow 1$. Analog sind die anderen Permutationen $1 \rightarrow 4$ (Sample 2), $4 \rightarrow 3$ (Sample 3) und $3 \rightarrow 2$ (Sample 4) zu betrachten.
Dieses Permutationsschema ist natürlich auch für höher dimensionierte Designs als zwei anwendbar. Im Folgenden (Tabelle 7.2) wird ein Design für drei Faktoren(X_1, X_2, x_3) und 100 Wertebereichsintervallen (Samples) vorgestellt.

Sample	X_1	X_2	X_3
1	25	13	74
2	14	91	7
...	...	...	...
39	47	32	56
...	...	...	...
100	69	4	84

Tabelle 7.2.: Eine $LHS_{3,100}$ Design-Matrix für m=3 Variablen und n=100 Samples

Auch hier ist wieder erkennbar, dass eine einfache Permutation ausreicht, um das Design nach den LHS-Regeln zu erstellen. Bei der Umsetzung des LHS in dieser Projektgruppe wurde daher diese Eigenschaft dann auch ausgenutzt und mittels der GSL (GNU Scientific Library), im speziellen der Funktion `gsl_ran_shuffle`, realisiert.

Ein mögliches Problem könnte bei der zufälligen Verteilung der Design-Punkte natürlich gerade *diese* Zufälligkeit sein: Ist die Verteilung nämlich unzureichend „gut" über den gesamten zu untersuchenden Bereich verteilt (bspw. befinden sich alle Punkte auf der Hauptdiagonalen), stehen dem Predictor im folgenden keine sinnvollen Korrelationen zu Verfügung. Um das Beispiel mit der Hauptdiagonalen noch einmal aufzugreifen: In so einem Fall, wären die zwei der Hauptdiagonalen gegenüber liegenden Ecken des Suchbereichs völlig „verwaist" und folglich wäre die Response-Surface dort nur schlecht bwz. gar nicht approximierbar. Dieser `worst case` wird in der Realisierung dieser Projektgruppe durch entsprechende Abfragen jedoch ausgeschlossen. Davon abgesehen ist jedoch jede Streuung, die durch den GSL-Pseudo-Zufallszahlen-Generator erzeugt wird, möglich.

7.2.2. Interpolation mittels Kriging

In den nichtstatistischen Interpolationsverfahren wird der Wert der Beobachtungsvariable an einem unbeprobten Ort durch ein gewichtetes Mittel der benachbarten gemessenen Werte geschätzt. Auch im Interpolationsverfahren Kriging wird ein unbekannter Wert durch ein gewichtetes Mittel der bekannten Nachbarwerte geschätzt. Die Gewichte werden dabei über Korrelationen zwischen den Variablen berechnet. Eine hohe Korrelation zwischen zwei Faktoren bewirkt dabei einen hohen Grad bei der Gewichtung, eine niedrige Korrelation entsprechend ein niedrigeres Gewicht. Berechnung der Korrelationsmatrix nach Jones, Schonlau und Welch:

$$\mathbf{R} = (r_{ij}) \text{ mit } r_{ij} = e^{-d(\mathbf{x}^{(i)},\mathbf{x}^{(j)})}$$

Die im Exponenten nötige Berechnung der Distanzen zwischen zwei Punkten erfolgt über diese Zwischenrechnung:

$$d(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \sum_{l=1}^{k} \left(x_l^{(i)} - x_l^{(j)} \right)^2$$

Mit diesen berechneten Korrelationen, ist dann der Predictor komplett, die λ_i entsprechen dabei den Gewichten.

$$\hat{Y}(x_{n+1}) = \sum_{i=1}^{n} \lambda_i * Y(x_i)$$

Die durch den Predictor berechneten Punkte des Metamodells werden dann dem Metamodell hinzugefügt. So wirken sich also auch durch Interpolation errechnete Punkte auf neu zu berechnende Punkte aus. Natürlich kann sich durch diese „Weiterverwendung“ der Fehler in der Approximation erhöhen.

Aus diesem Fakt und der Berechnung der Gewichte durch die Korrelationsmatrix folgt, dass dieses Verfahren für Extrapolation -selbst in direkter Nähe von Design-Punkten-ungeeignet ist. Man stelle sich bspw. vor, dass man den Rand einer Response-Surface untersuchen möchte. Unglücklicherweise lieferte das LHS zuvor jedoch nur einen Design-Punkt; und diesen womöglich noch relativ weit entfernt von der zu approximierenden Stelle. Wollte man nun durch den Predictor an dieser Stelle die Response-Surface approximieren, steht dem Predictor nur die Korrelation des einen Design-Punkts mit einem unendlich weit entfernten (bzw. nicht existenten) Punkt zu Verfügung. Damit kann natürlich keine ordentliche Approximation erfolgen.

7.3. Approximation durch Kriging Metamodelle

7.3.1. Approximation „einfacher“ Funktionen

Nachdem nun in den ersten Kapiteln über die Technik von Kriging berichtet wurde, werden nun in diesem Kapitel die ersten Ergebnisse des Software-Entwurfs präsentiert. Zu Beginn wurde sich ausschließlich der (rauschfreien) Benchmark-Funktionen aus dem ersten Semester der Projekt-Gruppe bedient. Betrachten wir daher beispielhaft auf den Abbildungen 7.3 und 7.4, wie eine Approximation der Sphere-Funktion ($\sum_i^n X_i^2$)durch ein Kriging-Metamodell ausschaut.

Zu sehen ist in Abbildung 7.3 der reale Verlauf der Response-Surface der Sphere-Funktion($\sum_i^n X_i^2$) im Definitionsbereich [-2;+2] mit seinem Minimum bei (0/0). In Abbildung 7.4 hingegen ist die Approximation der Sphere-Funktion im gleichen Definitionsbereich aufgetragen. Diese Approximation basiert auf 10 Design-Punkten. Wie zu sehen ist, ist eine Abweichung des Metamodells vom Original nur an den Rändern vorhanden: Das Minimum, und die wesentliche Eigenschaft der Funktion (exponentielles Wachstum der Response-Werte) wurden korrekt wiedergegeben. Dass der Funktionsverlauf gut angenähert wird, ist besonders gut durch den Konturplot in der X-Z-Ebene von Abbildung 7.4 erkennbar: Hier sind geradezu konzentrische Kreise um den Punkt des Minimums (0/0) zu beobachten. Das bedeutet, dass in alle Richtungen weg vom Minimum der Anstieg der Response-Surface fast identisch schnell erfolgt.

In Abbildung 7.5 ist der absolute Fehler der Approximation der Sphere-Funktion zur realen Funktion gegeben. Es wurde dazu die Differenz aus dem realen Modell und der Approximation gebildet. Eine absolut korrekte Approximation würde also als Ergebnis

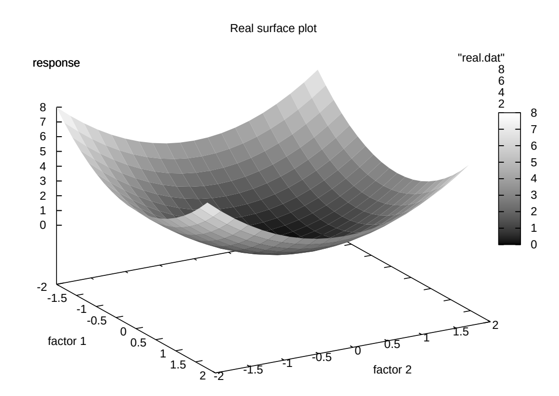



Abb. 7.3.: Reale Response-Surface der Sphere-Funktion für 2 Faktoren

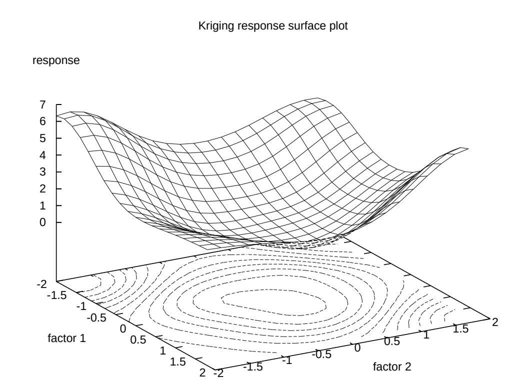


Abb. 7.4.: Durch Kriging approximierte Response-Surface der Sphere-Funktion für 2 Faktoren

hier eine plane Fläche auf Null-Niveau zeigen. Die zuvor subjektive Beurteilung der Approximation anhand des Konturplots des 10 Punkte-Beispiels wird hiermit nun bestätigt: Lediglich in einigen Randbereichen erhält man nicht die gewünschte Fläche auf der Nullebene, sondern einige kleine Äusreißernach oben bzw. nach unten.

Als abschließende Betrachtung dieses Beispiels wäre es nun interessant die Positon der Design-Punkte aus dem LHS zu kennen. Wie bereits erwähnt, wurden in diesem relativ großen Wertebereich (4X4) lediglich zehn Design-Punkte benötigt. Abbildung 7.6 zeigt das LHS-Design mit diesen zehn Punkten, deren Position und die resultierende Response-Surface als Konturplot.

Nach dem ersten Beispiel mit zehn Design-Punkten stellen sich nun folgende Fragen:

1. Wieviele Design-Punkte sind mindestens für eine „ordentliche" Approximation nötig?

2. Ab welcher Anzahl Design-Punkten führt eine Vergrößerung nur noch zu marginalen Verbesserungen im resultierenden Modell?

3. Wie geht das Design mit verrauschten / randomisierten Funktionen um bzw. ist bei Rauschen überhaupt eine Approximation der Funktion „sinnvoll" möglich?

Diese Fragestellungen allgemein zu beantworten ist natürlich aufgrund diverser Probleme unmöglich. Ein Problem, den ersten der drei Punkte betreffend, wäre z.B. die Untersuchung einer hoch-modalen Funktion (siehe 7.3.2). Das betrachtete Beispiel der Sphere-Funktion ist insofern für die Kriging-Interpolation „einfach" zu lösen, als das es nur ein Minimum besitzt, welches relativ einfach zu bestimmen ist. Ein weiteres, die Interpolation schwer beeinflussendes, Problem stellt das Rauschen dar.

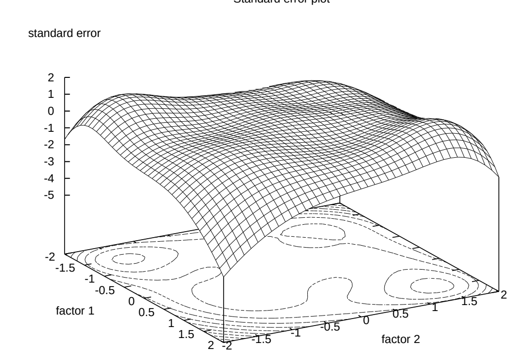


Abb. 7.5.: Absoluter Fehler der Approximation der Sphere-Funktion

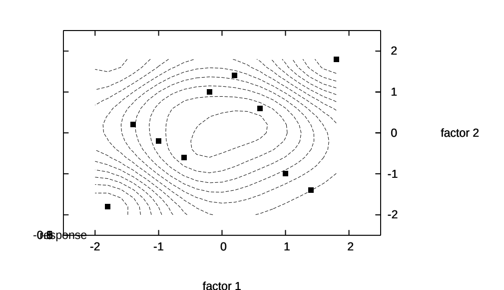


Abb. 7.6.: Design-Punkte aus $LHS_{2,10}$ Design-Matrix und resultierende Kontur der approximierten Response-Surface

7.3.2. Approximation einer hochmodalen Funktion

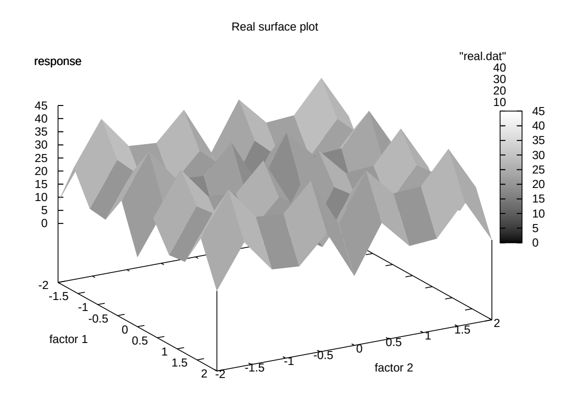
Betrachten wir als Beispiel für eine hochmodale und leicht verrauschte (1 % relatives Rauschen) Funktion die Rastrigin-Funktion

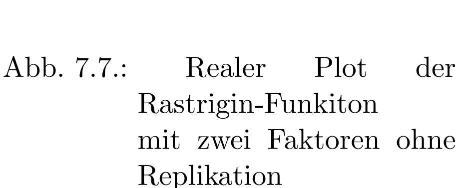
$$\sum_{i}^{n}(X_i^2 - 10cos(2\pi X_i) + 10)$$

Zu beachten ist bei dem Vergleich der beiden Plots in den Abbildungen 7.7 und 7.8, dass lediglich eine Auswertung der verrauschten Funktion durchgeführt wurde. Es wurden also keine Replikationen durchgeführt. Als Basis für die Approximation dienten in dieser Konstellation lediglich 25 Design-Punkte aus dem LHS.

Zu sehen ist in Abbildung 7.8 eine auf den ersten Blick „schlechte“ Approximation der Funktion durch das Kriging Metamodell. Dieses Verhalten ist auch intuitiv nachzuvollziehen: Woher soll der Predictor denn wissen, dass es eventuell zwischen zwei Punkten aus dem LHS ein Talmit einem lokalen Minimum gibt? Auf der anderen Seite: Sollte der Predictor rein zufällig just in einem dieser „Täler“ einen Design-Punkt finden, so wäre dieser Punkt natürlich dann zumindest ein lokales, bei keinem weiteren „Treffer-Glück“, sogar ein globales Minimum. Nun könnte natürlich argumentiert werden, dass diese relativ schlechte Approximation mit dem Fehlen von Replikationen oder einer zu geringen Anzahl von Design-Punkten (hier nur 25) zu begründen ist. Dies ist allerdings nicht der Fall, wie die Abbildungen 7.9 und 7.10 zeigen.

In diesem Fall wurden nun 50 Design-Punkte für die Approximation durch ein Kriging Metamodell jeweils zehn Mal ausgewertet und über diese zehn resultierenden Werte dann ein Durchschnitt gebildet. Dieser Durchschnittswert diente dann dem Predictor für die Interpolation als Berechnungsgrundlage. Aus den beiden Abbildungen ist keine




Abb. 7.7.: Realer Plot der Rastrigin-Funkiton mit zwei Faktoren ohne Replikation

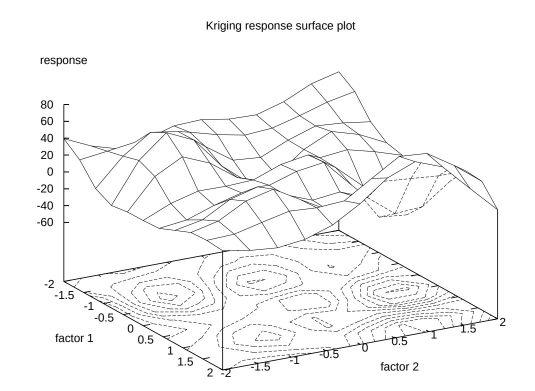


Abb. 7.8.: Durch Kriging approximierte Response-Surface der Rastrigin-Funktion für 2 Faktoren

signifikante Verbesserung in der Approximation zu erkennen. Als Fazit bleibt also zu sagen, dass die Approximation einer hochmodalen Funktion durch Kriging sehr schwer bzw. überhaupt nicht zu realisieren ist.

Allerdings will man ja im allgemeinen gar nicht die gesamte (=globale) Response-Surface des Modells nachbilden, sondern nur die für die Minimierung oder Maximierung nötigen Bereiche. Diese Motivation führte zu dem in Abschnitt 7.4.3 beschriebenen Optimierungs-Algorithmus namens „EGO-E“ (= efficient global optimization minus Effizienz).

7.4. Optimierung durch Kriging

7.4.1. Die Idee

Der einfachste Weg über eine Untersuchung der Response Surface ein Minimum zu finden, ist eine Approximation dieser Response Surface durch ein Meta-Modell und dann auf diesem Metamodell zu suchen. Wie aber bereits im vorherigen Abschnitt erwähnt, kann es grade bei hochmodalen Funktionen geschehen, dass man sich in einem lokalen Minimum „verfängt“ und es irrtümlich als globales Minimum betrachtet. Betrachten wir nun im Folgenden die in Abbildung 7.11 dargestellte Funktion: Die gepunktete Linie stellt das Ergebnis des Predictors dar und die durchgezogene Linie die (unbekannte) tatsächliche Funktion.

Es ist zu erkennen, dass der Predictor im Bereich des lokalen Minimums (x = 2,8) fast das exakte Minimum gefunden hat. Durch weitere Auswertungen in diesem Bereich wäre es ein Leichtes, den genauen Punkt des Minimums zu bestimmen. Allerdings wird bei dieser Art des Vorgehens das globale Minimum (ca bei x = 8.8) nicht gefunden, da wir uns nicht mehr aus dieser lokalen „Senke“ hinaus bewegen.

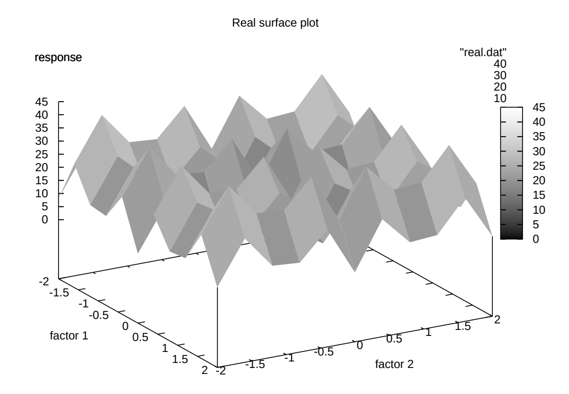


Abb. 7.9.: Realer Plot der Rastrigin-Funktion mit zwei Faktoren und 10 Replikation

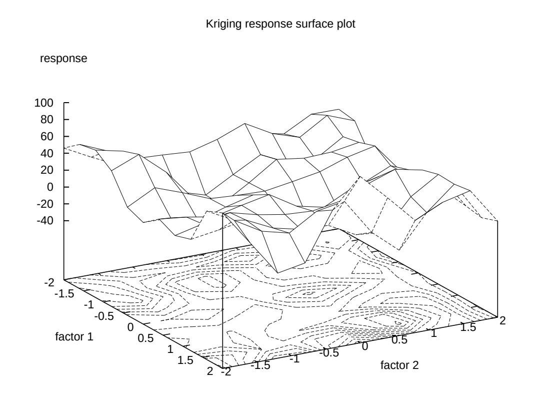


Abb. 7.10.: Durch Kriging approximierte Response-Surface der Rastrigin-Funktion für 2 Faktoren bei 10 Replikationen

Das Problem dieses Vorgehens ist also die Tatsache, dass eine möglicherweise große Unsicherheit des Predictors (ein großer Zwischenraum zwischen zwei Design-Punkten) nicht beachtet wird. Es wird also zu sehr auf dem bestehenden Meta-Modell gesucht, anstatt auch unsichere Bereiche (Bereiche mit wenigen Design-Punkten) zu untersuchen, in denen unter Umständen ein globales Optimum gefunden werden könnte (wie es ja in Abbildung 7.11 bei x = 8,8 zu erkennen ist).

Abbildung 7.12 zeigt den Standard-Fehler des Predictors. Bei dieser Abbildung gibt es beispielhaft zwei interessante Bereiche: Zum einen den in der Nähe von x = 2 und zum anderen der Bereich zwischen x = 4 und x = 12.

Warum sind für uns gerade diese Bereiche interessant? Die Antwort darauf ist offensichtlich, denn wir können beide „Extrema“ unserer Approximation betrachten: Zum

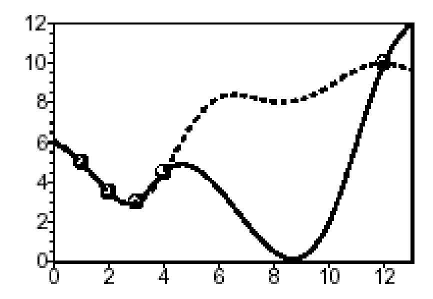


Abb. 7.11.: gepunktete Linie: Ergebnis des Predictors, durchgezogene Linie: Tatsächliches Modell

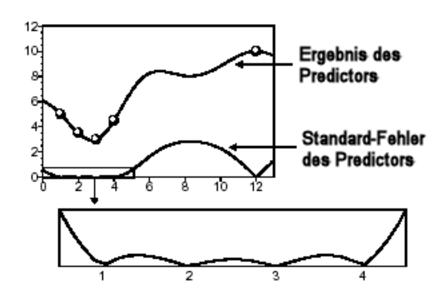


Abb. 7.12.: Der Standardfehler des Predictors bei einem simplen fünf-Punkte Datensatz

einen stehen im Bereich um x = 2 dem Predictor vier Design-Punkte innnerhalb eines recht kleinen Bereichs zu Verfügung (Bereich zw. 0 und 5 in Abbildung 7.12 unten vergrößert). Als Folge davon ist der Standard-Fehler in diesem Bereich sehr gering.

Das andere „Extremum“ ist der Bereich zwischen x = 4 und x = 12. In diesem großen Bereich haben wir keinerlei Design-Punkte auf die sich der Predictor stützen kann. Dies drückt sich für unsere Betrachtungen als ein großer Standard-Fehler aus. Sein Maximum erreicht der Fehler in diesem Bereich bei ca. x = 8,3. Ergo wäre diese Stelle ein viel versprechender Punkt für eine Suche nach einem globalen Minimum und eine Auswertung des Simulationsmodells wäre sinnvoll.

Allerdings würde es keinen Sinn machen nun die Suche nach dem globalen Optimum nur auf dieses Vorgehen zu stützen. Denn in dem Falle würde sich der Algorithmus dann nur noch auf die Unsicherheit des Modells konzentrieren und bereits gefunden eventuelle (lokale) Minima außer Acht lassen. Es ist nun die Aufgabe zu lösen, einen Mittelweg zwischen der Betrachtung der lokalen Minima und den unsicheren Bereichen zu finden.

Eine Möglichkeit für diesen Mittelweg stellt das sogenannte *Expected Improvement* (Kapitel 7.4.2) dar.

7.4.2. Expected Improvement

Die Idee und die Berechnung des expected improvements - also der erwarteten Verbesserung des Metamodells an einer bestimmten Stelle - lautet wie folgt:

Gehen wir davon aus, dass unser bisheriges Optimum bzw. der beste Funktionswert $f_{min} = \min(y^1, \ldots, y^n)$ ist. Bevor wir an einer weiteren Stelle x einen Simulationsaufruf starten, besitzen wir keinerlei Kenntnis über den Funktionswert $y(x)$. Natürlich ist an $y(x)$ nichts zufälliges, sondern der Wert ist uns einfach nur nicht bekannt. Modellieren wir nun diese „Unkenntnis“ bzw. „Unsicherheit“ durch eine normalverteilte Zufallsvariable mit dem Mittelwert und der Standard-Abweichung, die der Predictor bereitstellt (Predictor-Ergebnis an dieser Stelle).

Diese Idee wird in Abbildung 7.13 illustriert: Es wird an der Stelle mit dem größten Standardfehler die Normalverteilungskurve einer Zufallsvariablen Y eingezeichnet mit dem Mittelwert, den uns zuvor der Predictor als Interpolation für das Metamodell an dieser Stelle (x = 8) berechnet hat. Betrachtet man nun diese Zufallsverteilung von Y, so stellt man leicht fest, dass es sehr wahrscheinlich ist, dass man an dieser Stelle einen womöglich besseren Funktionswert erhält als den, den zuvor der Predictor berechnet hat. Das ist genau dann der Fall, wenn die Dichtefunktion von Y *unter* der Funktionskurve noch breite Ausläufer besitzt. Verschiedene untersuchte Stellen besitzen natürlich unterschiedliche Dichtefunktionen (siehe Berechnung der Zufallszahlenverteilung oben).

Durch dieses Modell erhält man also eine Aussage über die Wahrscheinlichkeit, an dieser bestimmten Stelle einen besseren Funktionswert als den bis dato besten (f_{min}) zu erhalten.

Formal betrachtet ist die Verbesserung an einer Stelle X: $I = max(f_{min} - Y, 0)$. Dieser Ausdruck für I ist eine Zufallsvariable, da auch das benutzte Y eine Zufallsvariable ist (es modelliert unsere „Unsicherheit“ über den Funktionswert an der Stelle x). Um den

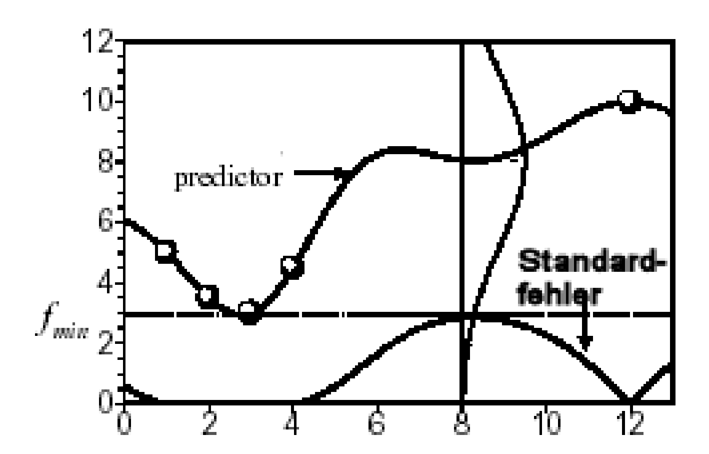


Abb. 7.13.: Die Unsicherheit über einen Funktionswert an einer Stelle (hier für x = 8) kann aufgefasst werden, als wäre er eine normalverteilte Zufallsvariable mit einem Durchschitt und einer Standardabweichung – gegeben vom Predictor – und des Standard-Fehlers an dieser Stelle (siehe Kapitel 7.4.1)

expected Improvement zu erhalten, muss nur der erwartete Wert berechnet werden:

$$E[I(x)] \equiv E[max(f_{min} - Y, 0)]$$

Um diesen Erwartungswert zu berechnen, werden nun die zuvor erwähnten Charakteristika der Normalverteilung (erhalten vom Predictor an der zu untersuchenden Stelle) durch $\hat{y}$ und s verkürzt. In dieser Notation ist Y Normal$(\hat{y}, s^2)$. Nach einigen mühsamen Partialintegralen lässt sich die rechte Seite der gerade genannten Fomel dann auf folgende (implementierbare) Form bringen:

$$E[I(x)] = (f_{min} - \hat{y})\Phi\left(\frac{f_{min} - \hat{y}}{s}\right) + s\phi\left((\frac{f_{min} - \hat{y}}{s}\right)$$

In dieser Formel stehen $\phi(.)$ bzw. $\Phi(.)$ für die Standardnormale Dichte- bzw. deren Verteilungsfunktion (zu beachten ist außerdem, dass es sich um s, und nicht um s^2 handelt).

Betrachten wir nun als Beispiel für den expected Improvement die triviale eindimensionale Zielfunktion in Abbildung 7.14. Überraschenderweise stellen wir in Abbildung 7.14(a) fest, dass diese Funktion zwei Peaks liefert, einen bei $x = 2,8$ und einen weiteren bei $x = 8,3$. Der erste Peak bei $x = 2,8$ ist größer, folglich führen wir an dieser Stelle (in der Nähe des bisherigen [eventuell nur lokalen] Minimums) einen Simulationslauf durch. In der nächsten Iteration (Abbildung 7.14(b)) stellen wir jedoch fest, dass der expected Improvement an der Stelle $x = 8,8$ maximal ist, und wir suchen in diesem Fall global weiter nach einem Minimum.

Wie bereits dieses kleine und anschauliche Beispiel aufzeigt, ist die expected Improvement-Funkion hochgradig multimodal. Es ist einfach zu zeigen, dass die expected

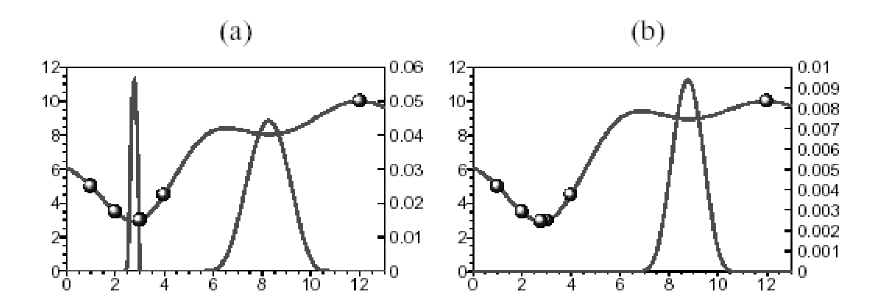


Abb. 7.14.: (a) Expected Improvement-Funktion nachdem lediglich fünf Punkte ausgewertet wurden; (b) der expected Improvement nachdem an der Stelle x = 2,8 ein Punkt hinzugefügt wurde [jeweils in (a) und (b) linke Skala für die Zielfunktion und die rechte Skala für den expected Improvement]

Improvement-Funktion bei ausgewerteten Punkten gleich Null und positiv zwischen diesen Punkten ist. Wie auch in diesem Beispiel zu sehen, kann es große Bereiche geben, in denen die erwartete Verbesserung gleich Null (und damit sehr „flach") ist. All dies macht die Optimierung der expected Improvement-Funktion recht schwierig. Jedoch ist die oben aufgezeigte Darstellung der expected Improvement-Funktion in geschlossener Form vorhanden und daher mit Rechnerunterstützung nutzbar. Die darauf basierende Implementierung wird in Kapitel 7.4.3 ausführlich beschrieben.

7.4.3. Der EGO-E-Algorithmus

7.4.3.1. Beschreibung der Funktionalität

Wie könnte nun ein Algorithmus ausschauen, der zum einen die expected Improvement-Funktion optimiert und damit einhergehend einen Optimierer für das gesamte Metamodell darstellt? Die Grundidee für den von den Autoren „EGO-E-Algorithmus" genannten Algorithmus ist eine Variante eines Divide-and-conquer-Algorithmus. EGO-E steht dabei für „efficient global optimization MINUS Effizienz". Der Name resultiert aus dem EGO-Algorithmus, wie er bspw. in [1] beschrieben wird, und einigen Modifikationen dieses Algorithmus auf Autoren-Seite, die dem Algorithmus ein geringes Maß an Effizienz kosten, zugleich allerdings eine erheblich einfachere Implementierung zur Folge haben.

Nun aber zum Vorgehen des EGO-E-Algos:

Zuerst führen wir eine Rasterung des Suchbereichs durch. Diese Rasterung ist völlig unabhängig von der Rasterung wie sie im LHS benutzt wird. Sie kann allerdings die gleich Schrittweite besitzen. Die Schrittweite gibt an, wie groß der Abstand zwischen zwei „Kreuzungen" des Rasternetzes ist. Abkürzend wird sie im Folgenden mit s bezeichnet. Sie berechnet sich durch

$$s = \frac{\text{Gesamtbreite des Wertebereichs}}{\text{Rastersteps}}.$$

Rastersteps steht dabei für die Anzahl der vorzunehmenden Unterteilungen und kann dem Optimierer als Parameter mitgeteilt werden.

Nun, da das Raster auf dem Suchbereich steht, untersuchen wir auf jeder „Kreuzung“ (= Schnittpunkt zweier Rasterlinien) den Wert des expected Improvement der entsprechenden Koordinaten (= Variablenbelegungen für die Faktoren). Dann werden all diese berechneten Werte verglichen und der *maximale* expected Improvement-Wert bestimmt. An der Stelle, an der die erwartete Verbesserung des Metamodells maximal ist, werten wir das Simulationsmodell aus, führen also eine (eventuell teure) Simulation durch. Nach Auswertung dieser Stelle wird dieser neue Punkt dem Design hinzugefügt und der Predictor bezieht ihn bei der neuen Berechnung des Metamodells mit ein. Anschließend wird wieder für jede „Kreuzung“ dessen expected Improvement berechnet und dann an der Stelle mit dem maximalen expected Improvement ein Simulationsaufruf gestartet. Dieses Vorgehen wird solange beibehalten, bis entweder:

a) ein Grenzwert für den Wert des maximalen expected Improvements erreicht wird ($\rightarrow Terminierung$), oder

b) eine bestimmte Anzahl von Iterationen lang kein besserer Funktionswert als der bis dato gefundene erreicht wird.

Während die Bedingung a) eine Terminierung des Algorithmus zur Folge hat, ist das weitere Vorgehen für Fall b) etwas komplizierter: In diesem Fall wird nämlich rund um den aktuell besten Punkt eine Verkleinerung des Suchbereichs durchgeführt. Die Abmessungen sind einfach berechnet: In jeder Koordinatendimension wird ein Schritt der Länge s gegangen (Definition s s.o.). Im zweidimensionalen Fall ergäbe dies dann folglich ein Quadrat der Fläche $4s^2$. Innerhalb dieses kleineren Suchbereiches wird dann wiederum eine Rasterung durchgeführt. Der Parameter „Rastersteps“ bestimmt auch hier wieder die Anzahl der Unterteilungen des Suchbereichs. Da der Suchbereich nun insgesamt um ein Vielfaches kleiner ist, als der bisherige Suchbereich, ist auch die Variable s neu zu berechnen und zu aktualisieren. Nachdem dies nun geschehen ist, greift wieder das gleiche Vorgehen wie zuvor: Berechnung aller expected Improvement-Werte und darunter dann die Bestimmung des maximalen expected Improvement-Wertes. An dessen Stelle wird dann das Simulationsmodell ausgewertet und das Metamodell aktualisiert.

Dieses Verkleinern des Suchbereichs geschieht solange, bis entweder:

a) ein Grenzwert für den Wert des maximalen expected Improvements erreicht wird ($\rightarrow Terminierung$), oder

b) eine zuvor festgelegte Anzahl von Verkleinerungen bereits durchgeführt wurden.

Es existieren also zwei voneinander unabhängige Terminierungsmöglichkeiten: Zum einen das Erreichen eines Grenzwertes für den expected Improvement-Wert, ab dem eine weitere Auswertung des Simulationsmodells nicht mehr eine genügend große Wahrscheinlichkeit für eine Verbesserung verspricht, und zum anderen eine Anzahl an Verkleinerungen. Beide Terminierungsarten können durch den Benutzer eingestellt werden. Während der Studien des Algorithmus haben sich jedoch für den Grenzwert 0,001 und für die Anzahl der Verkleinerungen 3 als sinnvoll herausgestellt. Für den Parameter, der angibt, nach wievielen Simulationsauswertungen ohne Verbesserung eine Suchbereichs-Verkleinerung durchgeführt wird, hat sich hingegen 20 als gut erwiesen.

Sample	X_1	X_2	Response-Wert	best	Max. exp. Improvement
1	-0,8	0,4	**0,8**	1,78	0,84724
2	0,6	-1	1,36	**0,8**	0,70968
3	$2,98 * 2^{-8}$	0,4	**0,16**	0,8	0,72338
4	-0,2	-0,4	0,2	0,16	0,66649
5	0,5	-0,2	0,29	0,16	0,81011
...	...	...	...	...	...
21	0,1	$2,98 * 2^{-8}$	**0,01**	0,1	0,26657
22	1,3	-1,3	3,38	**0,01**	0,28429
23	-1,5	$2,98 * 2^{-8}$	2,25	0,01	0,20664
24	0,2	-0,5	0,29	0,01	0,17688
...	...	...	...	...	...
41	$2,98 * 2^{-8}$	0,1	0,01	0,01	0,0208
42	$2,98 * 2^{-8}$	-0,01	$9,99 * 2^{-5}$	0,01	0,01092
43	0,035	-0,1	0,011	$9,99 * 2^{-5}$	0,00070

Tabelle 7.3.: Optimierungsablauf am Beispiel der Sphere-Funktion

7.4.3.2. EGO-E an einem Beispiel

In Tabelle 7.3 ist in Auszügen der Ablauf des in Kapitel 7.4.3.1 vorgestellten Algorithmus eingetragen. Als zu untersuchende Funktion wurde die Sphere-Funktion gewählt. Das Metamodell dieser Funktion basiert auf zehn Design-Punkten aus dem LHS im Wertebereich zwischen -2 und 2 für beide Faktoren X_1 und X_2. Desweiteren wurden folgende Einstellungen verwendet:

1. Rastersteps = 40
2. Zyklen ohne Verbesserung bis der Suchbereich verkleinert wird: 20
3. untere Schranke für maximalen expected Improvement: 0,001
4. Anzahl Suchbereichsverkleinerungen: 3

Mehrere Charakteristika des Algorithmus kann man in der Tabelle wieder finden. Dazu gehört zum Beispiel die Aktualisierung des bisher besten Response-Wertes durch einen soeben gefundenen besseren. Zu sehen ist dies zum einen bei den Schritten 1 $\rightarrow$ 2,2 $\rightarrow$ 3 sowie bei 21 $\rightarrow$ 22. Interessanter ist jedoch die Verfeinerung des Suchbereichs nachzuvollziehen. Dies geschieht nach Schritt 41. Zu diesem Zeitpunkt hat sich nämlich 20 Schritte lang der Response-Wert von 0,01 nicht verbessert. Aus den oben genannten Einstellungen der Parameter für dieses Beispiel bedeutet das also, dass ab jetzt der Suchbereich nicht mehr die gesamte Response-Surface, sondern nur noch einen Ausschnitt davon, darstellen wird. Der Mittelpunkt des neuen Suchbereichs befindet sich an den Koordinaten mit dem besten Response-Wert, hier (0,1/$2,98 * 2^{-8}$), also sehr nahe am Minimum (0/0).

Die Terminierung des Algorithmus geschieht nach Iteration Nummer 43. In diesem Durchlauf beträgt der Wert des maximalen expected Improvement nur noch 0,0007, ist also kleiner als der in den Einstellungen erwähnte Wert von 0,001. Das bedeutet, dass der bis dahin beste gefunde Response-Wert (0,01) als Optimum betrachtet wird. Dieser Wert ist nah am tatsächlichen Optimum.

Da der Algorithmus durch Unterschreitung der Grenze von 0,001 terminiert, fällt der Parameter, der die maximale Anzahl von Verkleinerungen des Suchbereiches festlegt, nicht ins Gewicht.

Zusammenfassend lässt sich für dieses Beispiel sagen, dass zwar nicht das tatsächliche Optimum 0 bei (0/0) gefunden wurde, sondern lediglich 0,01. Allerdings muss auf der anderen Seite auch bedacht werden, dass es lediglich 53 Auswertungen (10 Design-Punkte aus dem LHS und 43 Iterationen des EGO-E-Algos) des Simulationsmodells bedurfte, um diesen schon recht guten Wert zu erhalten. Die Interpolation mittels Kriging stellt also bisweilen eine echte Alternative zu anderen Verfahren wie z.B. der Response-Surface-Methode dar.

7.5. Abschlussbetrachtung

Was lässt sich nun rückblickend über Kriging als Approximations- und desweiteren als Optimierungsmethode sagen? Im allgemeinen stellt Kriging eine echte Alternative zu den klassischen Verfahren wie z.B. Response Surface Methode (RSM) oder den evolutionären Algorithmen dar. Es ist aufgrund ausgeklügelter mathematischer Verfahren möglich, aus den Response-Werten weniger Faktor-Kombinationen die Response-Werte anderer Faktor-Kombinationen zu errechnen bzw. zu interpolieren. Auch das Design hinter Kriging (Space-filling-Design, hier LHS) ist sehr flexibel und gestattet dem Benutzer zu entscheiden, mit wie vielen Design-Punkten als Grundlage die Interpolationen erfolgen soll. Dass diese Interpolationen nicht immer zu 100% dem tatsächlichen Simulationsmodell entsprechen, ist offensichtlich. Es ist allerdings bemerkenswert, wie gut sie in den meisten Fällen dieses Modell annähern können. Die Ersparnis von eventuell vielen sehr teuren (= zeitintensiven) Simulationsläufen ist ein großer Vorteil von Kriging.

Problematisch ist jedoch der Umgang mit multimodalen Simulationsmodellen. Anhand der Rastrigin-Benchmark-Funktion wurde beispielhaft aufgezeigt, daß eine Interpolation 7.10 einer hochgradig multimodalen Funktion sehr schwierig ist. Dies ist intuitiv nachvollziehbar und genauso offensichtlich ist es, dass in solchen Fällen lediglich eine massive Erhöhung der Anzahl der Design-Punkte zu einer tatsächlichen Verbesserung der Approximation führen kann. Damit einhergehend wäre dann allerdings der große Vorteil, den Kriging bietet, wieder hinfällig. Denn wenn bei der Interpolation nicht auf eine große Menge Simulationsauswertungen verzichtet werden kann, so wird die gesamte Idee ad absurdum geführt.

Teil II.

Simulation

8. Simulationswerkzeuge

8.1. Modellierung von Prozeßketten mit dem ProC/B Toolset

Jan Kriege

8.1.1. Einleitung

Das ProC/B-Toolset ist eine Sammlung von Werkzeugen zur Modellierung und Analyse logistischer Netze. Diese logistischen Netze lassen sich durch eine Anzahl von Unternehmen charakterisieren, die ein Netzwerk aus Auftraggebern und Anbietern bilden. Die Aktivitäten der Produktions- und Transportprozesse lassen sich in verschiedene Unteraktivitäten aufteilen, die wiederum verschiedenen Firmen und Abteilungen zugewiesen werden.
Das ProC/B-Toolset bietet die Möglichkeit, diese logistischen Netze grafisch zu modellieren und zu analysieren. Das Toolset wird im Rahmen des Sonderforschungsbereichs 559 "Modellierung großer Netze in der Logistik" entwickelt.
In der Logistik werden Systemabläufe üblicherweise durch Prozeßketten beschrieben. Dabei durchläuft ein Prozeß nacheinander festgelegte Aktivitäten, die Ressourcen anfordern. Folglich benutzt auch der ProC/B-Modellformalismus Prozeßketten, um diese Abläufe zu beschreiben und zum Beispiel nicht etwa Petri-Netze, die den Gewohnheiten in der Logistik nicht entsprechen.
Der ProC/B-Formalismus basiert auf einem Prozeßketten-Paradigma von A. Kuhn vom Fraunhofer-Institut für Materialfluß und Logistik.
Im folgenden Abschnitt werden ausführlich die Modellierung von Prozeßketten mit dem Prozeßketteneditor und die einzelnen Elemente einer Prozeßkette vorgestellt. Der letzte Abschnitt gibt einen Überblick über das ProC/B-Toolset und beschreibt kurz die Analysetools.

8.1.2. ProC/B-Formalismus

Wie bereits in der Einleitung erwähnt, bestehen logistische Netze aus einer strukturellen Hierarchie von Unternehmen und Abteilungen. Zusätzlich bilden die Aktivitäten bzw. Produktionsprozesse eine Verhaltens-Hierarchie: Die Aktivitäten lassen sich in Unteraktivitäten aufspalten, deren Unterteilung sich noch weiter verfeinern läßt. In der Praxis sind diese beiden Hierarchien miteinander verknüpft, da die Unteraktivitäten von Abteilungen der Unternehmen ausgeführt werden.
Beide Arten von Hierarchien lassen sich mit dem ProC/B-Formalismus darstellen: Die

strukturelle Hierarchie läßt sich über Funktionseinheiten (FEs) ausdrücken, die selbst wieder weitere FEs enthalten können. Das Verhalten wird durch Prozeßketten (PKs) modelliert.
Funktionseinheiten bieten Dienste an, die von ihrer Umgebung genutzt werden können. Diese Dienste werden im Inneren der FE durch Prozeßketten beschrieben. Prozeßketten wiederum können die Dienste einer FE nutzen, um ihre Aktivitäten auszuführen. Die Erzeugung und Beendigung von Prozeßketten wird durch Quellen und Senken spezifiziert. Die FE-Hierarchie beginnt also mit Quellen und Senken, enthält eine beliebige Anzahl selbstdefinierter FEs und endet mit Standard-Funktionseinheiten wie Counter, Server oder Storage.
Abbildung 8.1 zeigt eine einfache Hierarchie aus Funktionseinheiten, wie sie im Hauptfenster des Prozeßketteneditors dargestellt wird.

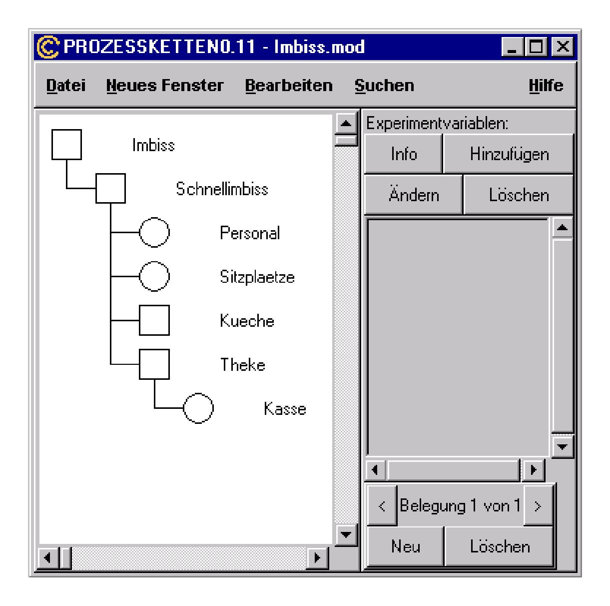


Abb. 8.1.: Darstellung der FE-Hierarchie im Prozeßketteneditor

Die Darstellung eines Modells im Prozeßketteneditor zeigt Abbildung 8.2. Diese Darstellung ist dreigeteilt: Der obere Bereich ist für die Namensgebung und eine eventuelle Parametrisierung gedacht. Der mittlere Teil beschreibt das Verhalten des Modells mit Hilfe von Prozeßketten, Quellen und Senken. Der untere Bereich dient zur Darstellung der strukturellen Hierarchie: Hier werden die enthaltenen FEs angezeigt. Zusätzlich können auch Variablen definiert werden.
Außerdem enthält die Darstellung noch Informationen darüber, welche Dienste der FEs von den Prozeßketten genutzt werden.

8.1.2.1. Prozeßketten

Prozeßketten bilden das Verhalten eines bestimmten Prozeß-Typs ab. Eine Prozeßkette besteht im Wesentlichen aus Prozeßkettenelementen (PKEs) und Konnektoren. Die

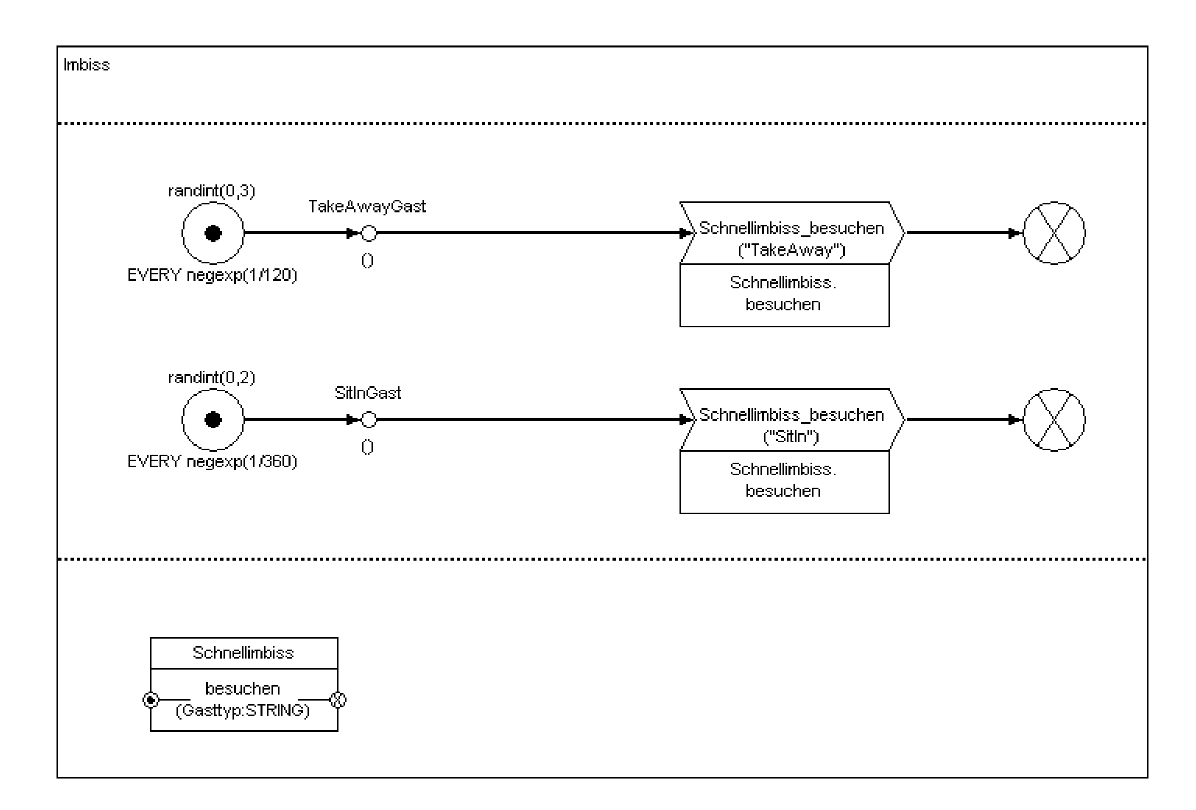


Abb. 8.2.: Darstellung eines Modells im Editor

einzelnen Elemente der PK sind geordnet. Durch Quellen und Senken (dargestellt durch die Kreise am Anfang und Ende der Prozeßkette in Abbildung 8.3) können Beginn und Beendigung von Prozessen spezifiziert werden.

Ein gestarteter Prozeß durchläuft nacheinander die einzelnen PKEs, an Konnektoren kann es zu Verzweigungen einzelner Teilketten kommen. Die Reihenfolge der Elemente wird durch Pfeile visualisiert, die die einzelnen Prozeßkettenelemente miteinander verbinden. Beim Durchlaufen der einzelnen Elemente kann, je nach Element, Zeit verbraucht werden, der Übergang von einem Element zum nächsten benötigt dagegen keine Zeit. Jede Prozeßkette verfügt neben den eigentlichen Elementen noch über eine ProzeßID ('PK1' und 'PK2' in Abbildung 8.3). Die ProzeßID legt den Namen der PK fest. Außerdem können über die ProzeßID Variablen und Parameter festgelegt werden. Im Gegensatz zu den globalen Variablen, die im unteren Bereich der Funktionseinheit angezeigt werden, gelten diese Variablen jeweils nur lokal für einen gestarteten Prozeß. Um zwischen den globalen und lokalen Variablen unterscheiden zu können, ist es nötig, die lokalen Variablen bei Benutzung mit dem Präfix 'data.' zu versehen.
PKEs werden im Editor als sechseckige Symbole dargestellt (zum Beispiel 'Element11' in Abbildung 8.3). PKEs dienen zum Aufruf der Dienste von Funktionseinheiten oder können als Delay-PKE eingesetzt werden. In Abbildung 8.3 ruft zum Beispiel 'Element11' den Dienst 'request' von 'Server1' mit dem Parameter 'negexp(9)' auf (negexp(9) ist die

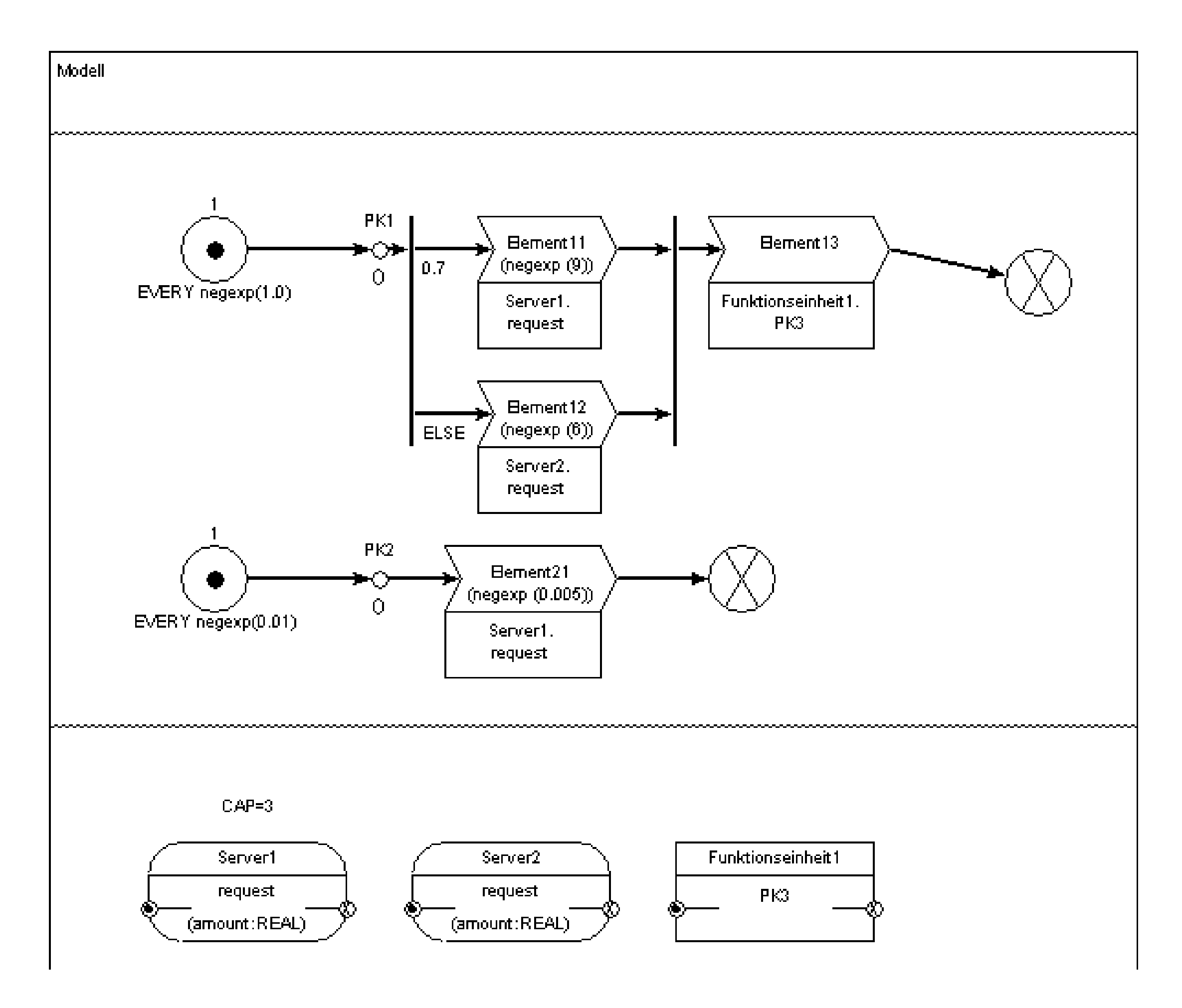


Abb. 8.3.: Darstellung eines einfachen Modells

Exponentialverteilung mit Parameter 9). Die benötigte Zeit richtet sich in diesem Fall nach dem aufgerufenen Dienst. Ein Delay-PKE dagegen führt keine Aktionen aus, sondern gibt lediglich eine Zeitspanne an, die vergeht bevor der Prozeß zum nächsten Element fortschreiten kann. Neben diesen beiden PKE-Typen gibt es noch Code-Elemente, die zum Beispiel benutzt werden können, um Variablen zu verändern.
Konnektoren dienen zur Aufteilung, Zusammenführung und Synchronisation von Prozessen. Der ProC/B-Formalismus sieht Oder-, Und- sowie PK-Konnektoren vor.
Ein öffnender und ein schließender Oder-Konnektor sind in Abbildung 8.3 als zwei vertikale Linien dargestellt. Oder-Konnektoren teilen die Prozeßkette in alternative Zweige auf. An den ausgehenden Kanten des öffnenden Oder-Konnektors wird angegeben, unter welchen Bedingungen die einzelnen Alternativen ausgewählt werden. Dies können Wahrscheinlichkeiten oder boolesche Ausdrücke sein. In der Abbildung 8.3 wird mit einer Wahrscheinlichkeit von 70% die obere Teilkette gewählt und mit einer Wahrscheinlichkeit von 30% die untere. Durch den schließenden Oder-Konnektor werden die alternativen Zweige wieder zusammengeführt.

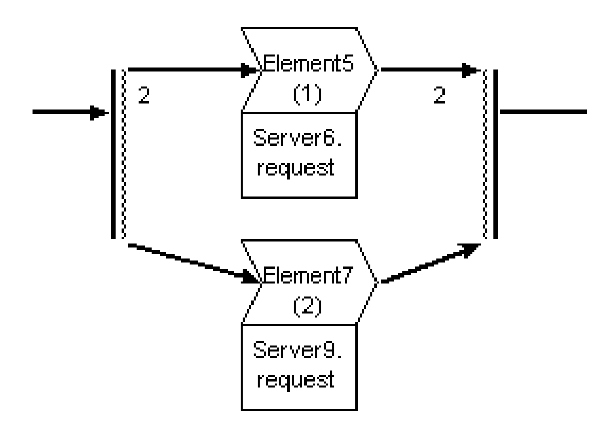


Abb. 8.4.: Ausschnitt einer PK mit Und-Konnektoren

Und-Konnektoren teilen eine Prozeßkette in mehrere Zweige auf, die parallel ablaufen. Abbildung 8.4 zeigt einen Ausschnitt einer Prozeßkette mit einem öffnenden und einem schließenden Und-Konnektor. Die Kantenbeschriftung drückt hier die Anzahl der gleichartigen Teilprozesse aus, in die der aktive Prozeß aufgeteilt werden soll. In dem Beispiel werden zwei Teilprozesse gestartet, die die Aktivität von 'Element5' ausführen und einer, der die Aktivität von 'Element7' ausführt. Der schließende Und-Konnektor dient auch hier wieder der Zusammenführung der Prozesse. Bevor der Gesamtprozeß weiter fortschreiten kann, wird am schließenden Und-Konnektor gewartet bis alle Teilprozesse beendet sind. Zusätzlich stellt der Editor noch einen Konnektor vom Typ 'Schließender und öffnender Und-Konnektor' zur Verfügung, der einen schließenden und einen darauffolgenden öffnenden Und-Konnektor vereint.

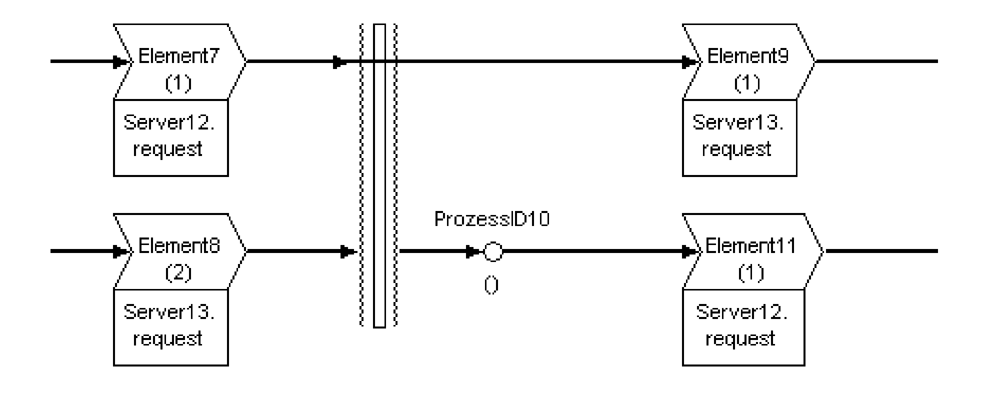


Abb. 8.5.: Ausschnitt eines Modells mit PK-Konnektor

Der PK-Konnektor dient der Synchronisation von Prozessen. Dabei werden mehrere

einfache Prozeßketten zu einer zusammengesetzten PK kombiniert. Abbildung 8.5 zeigt den Ausschnitt eines Modells mit PK-Konnektor. Alle eingehenden Kanten stehen für ankommende Prozesse. Wurden alle eingehenden Prozesse beendet, werden die ausgehenden gestartet. Über die Kantenbeschriftungen kann festgelegt werden, wieviele parallele Prozesse desselben Typs gestartet werden sollen oder beendet werden müssen. Eine Ausnahme von dieser Regel kann durch das Anlegen eines Durchgangsports erreicht werden. In Abbildung 8.5 sind die oberen einfachen PKs über den Durchgangsport miteinander verbunden, was durch die Linie durch den Konnektor angezeigt wird. Dadurch wird erreicht, daß der Prozeß individuell fortgeführt wird.

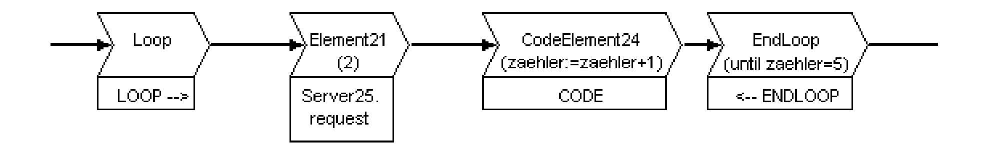


Abb. 8.6.: Ausschnitt eines Modells mit Loop/EndLoop-Elementen

Neben den Konnektoren stellt der ProC/B-Formalismus auch noch Loop-Elemente zur Verfügung, die zur Realisierung von Schleifen dienen. Der Teil der Prozeßkette, der zwischen dem Loop- und dem EndLoop-Element liegt, wird so lange ausgeführt, bis die angegebene Abbruchbedingung wahr wird. Ein einfaches Beispiel ist in Abbildung 8.6 dargestellt.

8.1.2.2. Quellen und Senken

Quellen und Senken spezifizieren, wie bereits erwähnt, die Entstehung und Beendigung von Prozessen. Der ProC/B-Formalismus kennt verschiedene Arten von Quellen und Senken: Unbedingte Quellen, wie zum Beispiel die Quellen in Abbildung 8.3, können zu einem bestimmten Zeitpunkt einen individuellen Prozeß starten oder jeweils nach einer bestimmten Anzahl Zeiteinheiten einen Prozeß initiieren. Zusätzlich kann angegeben werden, wie viele individuelle Prozesse jeweils gestartet werden sollen. Dadurch können gleichzeitig mehrere Prozesse aktiv sein, die von derselben Prozeßkette beschrieben werden.
Bei bedingten Quellen wird die Generierung der Prozesse von außen gesteuert, zum Beispiel durch Beendigung eines anderen Prozesses.
Unbedingte Senken beenden einen Prozeß, bei bedingten Senken wird zusätzlich aber noch ein neuer Prozeß einer anzugebenden Funktionseinheit gestartet.
Neben diesen beiden Arten von Quellen und Senken gibt es noch Virtuelle Quellen und Senken, die zusammen mit den Funktionseinheiten im nächsten Abschnitt besprochen werden.

8.1.2.3. Funktionseinheiten

Funktionseinheiten bilden die strukturelle Hierarchie eines logistischen Netzes ab. FEs führen bestimmte Aktivitäten aus, bieten also Dienste an, die innerhalb von Prozeßketten genutzt werden können.
Der ProC/B-Formalismus sieht vier Standard FE-Typen vor: Server, Prio-Server, Counter und Storage (siehe Abbildung 8.7).
Ein Server stellt eine Ressource zur Verfügung, die sich über den Dienst request für einen bestimmten Zeitraum anfordern läßt. Er ist zum Beispiel zur Darstellung einer Maschine geeignet. Für einen Server lassen sich die Bedien-Geschwindigkeit, die Bedien-Disziplinen und die Kapazität angeben. Die zur Verfügung stehenden Bedien-Disziplinen sind FCFS (First-Come-First-Serve), PS (Processor Sharing) und IS (Infinite Server). Bei FCFS werden die Anfragen in der Reihenfolge ihrer Beauftragungszeitpunkte bearbeitet. Übersteigt die Anzahl der Anfragen die Kapazität des Servers, müssen einige Anfragen warten bis Ressourcen frei werden. Das bedeutet, daß der Prozeß, von dem der Dienst request aufgerufen wurde, solange gestoppt wird. Bei der Bedienstrategie PS werden alle Anfragen gleichzeitig bearbeitet. Reicht die Kapazität des Servers nicht aus, verlängert sich also die Bearbeitungszeit für alle Anfragen. Bei IS werden ebenfalls alle Anfragen gleichzeitig bearbeitet. Der Server verfügt hier allerdings über eine unendliche Kapazität und kann beliebig viele Aufträge ohne Verzögerungen bearbeiten.
Der Prio-Server erfüllt dieselben Aufgaben wie der Server. Bei der Abarbeitung der Anfragen berücksichtigt er allerdings Prioritäten.
Ein Counter eignet sich zur Beschreibung eines Lagers oder zum Beispiel zur Erfassung einer begrenzten Anzahl von Arbeitsmitteln. Dazu verwaltet der Counter einen mehrdimensionalen Vektor, für den ein Initialisierungswert und eine Ober- und Untergrenze festgelegt werden. Der Initialisierungswert beschreibt die zu Anfang vorhandene Belegung des Counters. Über den Dienst change können Elemente angefordert oder freigegeben werden. Positive Werte des Paramaters für den Dienst change entsprechen Anforderungen, negative Werte Freigaben. Würde bei einer Anforderung die gegebene Obergrenze überschritten, bzw. die Untergrenze unterschritten, wird die Anforderung nicht durchgeführt. Der Prozeß stoppt so lange bis die Aktivität ohne Verletzung der Grenzen durchgeführt werden kann.
Ein Storage erweitert den Counter um zusätzliche Dienste, die helfen den aktuellen Zustand des Elements zu ermitteln. So kann zum Beispiel über den Dienst content die aktuelle Belegung des Storage abgefragt werden.

Zusätzlich zu den Standard-FEs können auch eigene Funktionseinheiten konstruiert werden. Diese FEs verfügen über eine Außen- und eine Innenansicht. Abbildung 8.8 zeigt einen Modelausschnitt mit der Außenansicht der FE 'Kunden'. In der Außenansicht werden die angebotenen Dienste (in diesem Beispiel der Dienst 'Bestellung') mit den notwendigen Parametern angegeben. Sofern der Dienst Werte zurückliefert, werden auch diese hier angezeigt. Am linken und rechten Rand der FE werden eine virtuelle Quelle und eine virtuelle Senke angezeigt. Über diese Quelle kann der Dienst mit einem aufrufenden Prozeßkettenelement verbunden werden.
Abbildung 8.9 zeigt die Innenansicht einer Funktionseinheit. Das innere der FE besteht

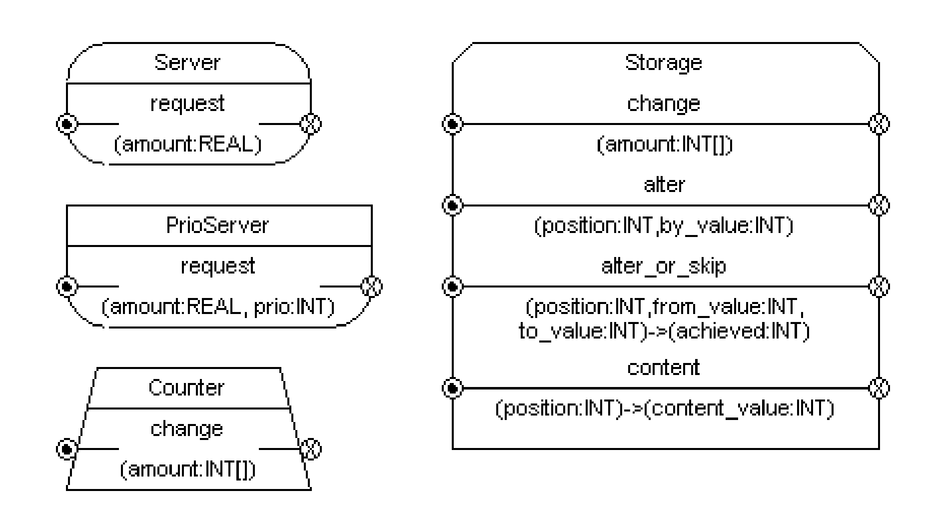


Abb. 8.7.: Standard FE-Typen: Server, Prio-Server, Counter, Storage

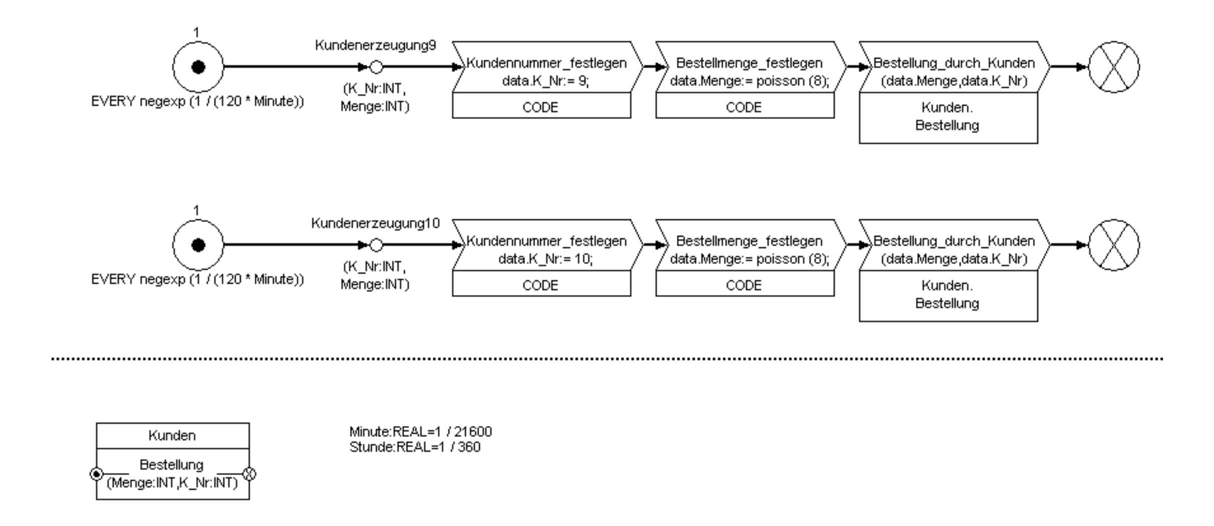


Abb. 8.8.: Modell mit Außenansicht einer Funktionseinheit

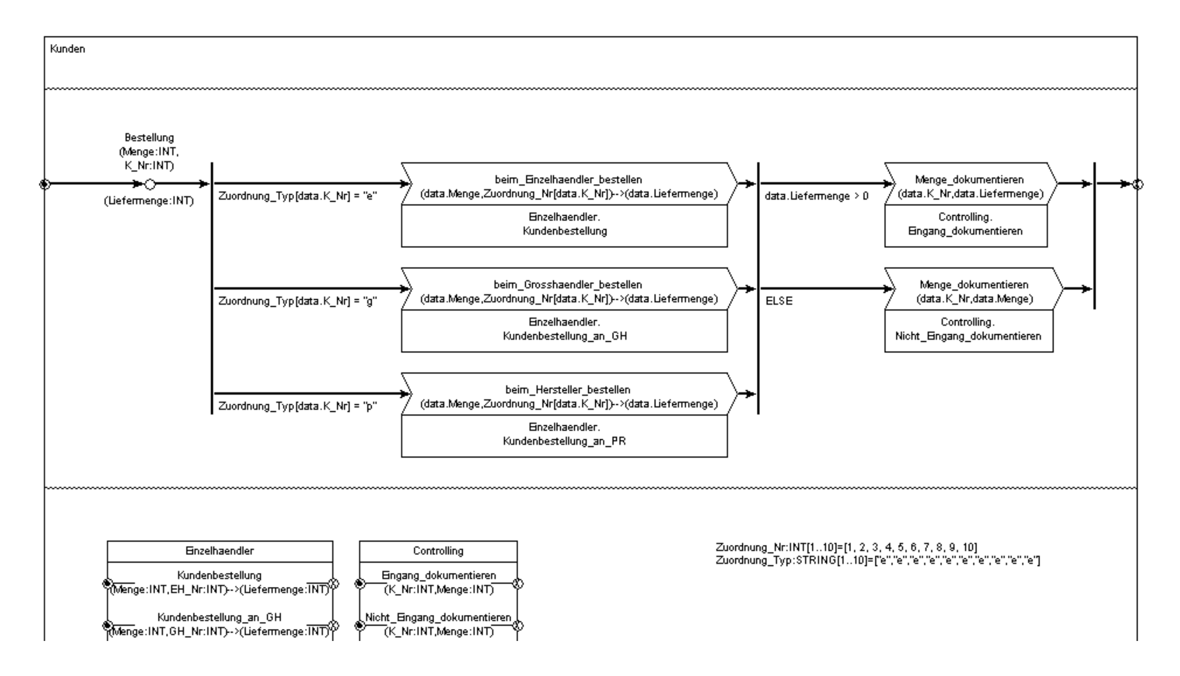


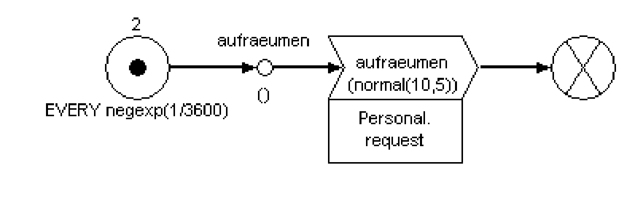
Abb. 8.9.: Innenansicht einer Funktionseinheit

wieder aus Prozeßketten, die das Verhalten beschreiben und eventuell weiteren FEs, von denen wiederum Dienste genutzt werden. Die enthaltenen Prozeßketten sind allerdings nicht mit unbedingten oder bedingten Quellen und Senken verbunden, sondern jeweils mit einer Virtuellen Quelle und einer Virtuellen Senke, die am linken und rechten Rand sichtbar sind. Dadurch wird gekennzeichnet, daß es sich bei diesen Prozeßketten um Dienste handelt, die die FE anbietet.

Eine Besonderheit bieten externe Funktionseinheiten. Bis jetzt griffen Prozeßkettenelemente immer nur auf Dienste von FEs zu, die in der selben Funktionseinheit liegen wie sie selbst. Es kann allerdings wünschenswert sein, auch auf Dienste von FEs zugreifen zu können, die in der Hierarchie an anderer Stelle auftauchen.
Abbildung 8.10 zeigt einen Ausschnitt aus einem Modell mit der Funktionseinheit 'Kueche' und dem Server 'Personal'. Innerhalb der FE 'Kueche' soll ebenfalls auf den Server zugegriffen werden. Dies kann durch eine externe Funktionseinheit erreicht werden. Eine Funktionseinheit verfügt über eine Prozeßzuordnungsliste, über die der externen Funktionseinheit Dienste anderer FEs zugewiesen werden können. Diese Prozeßzuordnung wird im unteren Bereich der Außenansicht dargestellt (Personal_request TO Personal.request). Im Inneren der Funktionseinheit steht nun eine externe Funktionseinheit mit dem festen Standardnamen EXTERNAL zur Verfügung, über die der Dienst des Servers genutzt werden kann (siehe Abbildung 8.11).

8.1.3. ProC/B-Toolset

Neben dem Editor zur Modellierung von Prozeßketten besteht das ProC/B-Toolset noch aus diversen weiteren Tools, die eine Simulation und Analyse der Modelle erlauben.



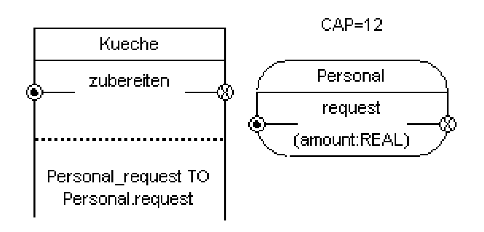


Abb. 8.10.: Außenansicht einer FE mit externer Funktionseinheit

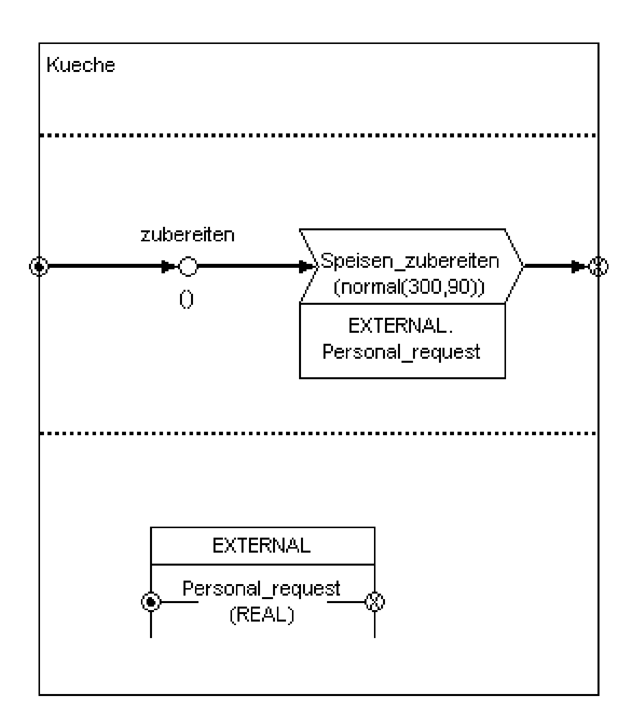


Abb. 8.11.: Innenansicht einer FE mit externer Funktionseinheit

Einen groben Überblick über das Toolset zeigt Abbildung 8.12.

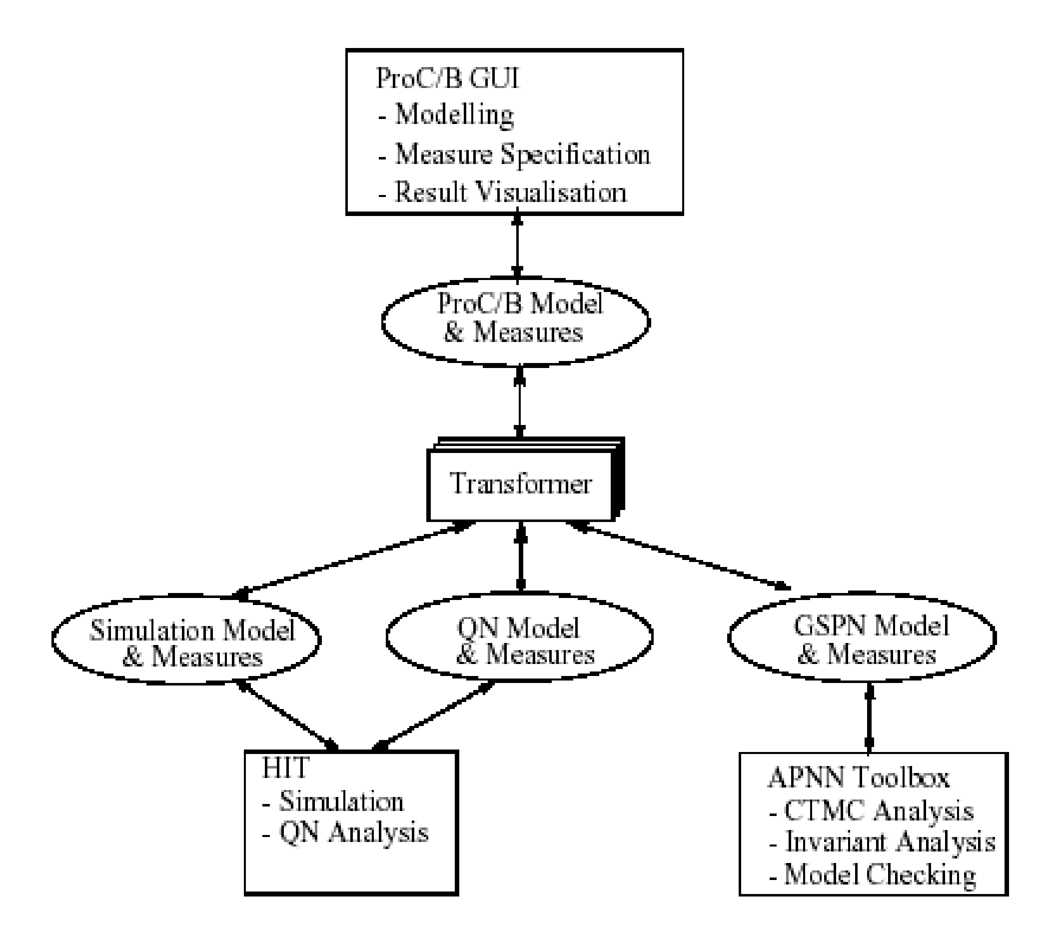


Abb. 8.12.: ProC/B-Toolset

Mit dem Prozeßketteneditor des ProC/B-GUI können, wie bereits ausführlich beschrieben, Modelle erstellt und Simulationen angelegt werden. Hier kann zum Beispiel festgelegt werden, von welchen Funktionseinheiten Auslastung etc. gemessen werden sollen. Zum ProC/B-Toolset gehören Konverter, die das Modell und die Experimentbeschreibung umwandeln, so daß die Beschreibungen von Analysetools genutzt werden können. Auf diese Art und Weise können die Modelle zum Beispiel mit HIT simuliert werden. Die Ergebnisse der Simulation lassen sich im ProC/B-GUI grafisch darstellen (Abbildung 8.13). Eine weitere Möglichkeit stellt die Konvertierung des ProC/B-Modells in Queueing Networks dar. Die QN-Analyse kann ebenfalls mit Hilfe von HIT durchgeführt werden.

Ebenfalls möglich ist eine Transformation der Modelle in eine Petri-Netz-Darstellung für die APNN-Toolbox, mit der eine numerische Analyse (Continous time Markov Chains), Analysen von Invarianten und Model Checking durchgeführt werden können.

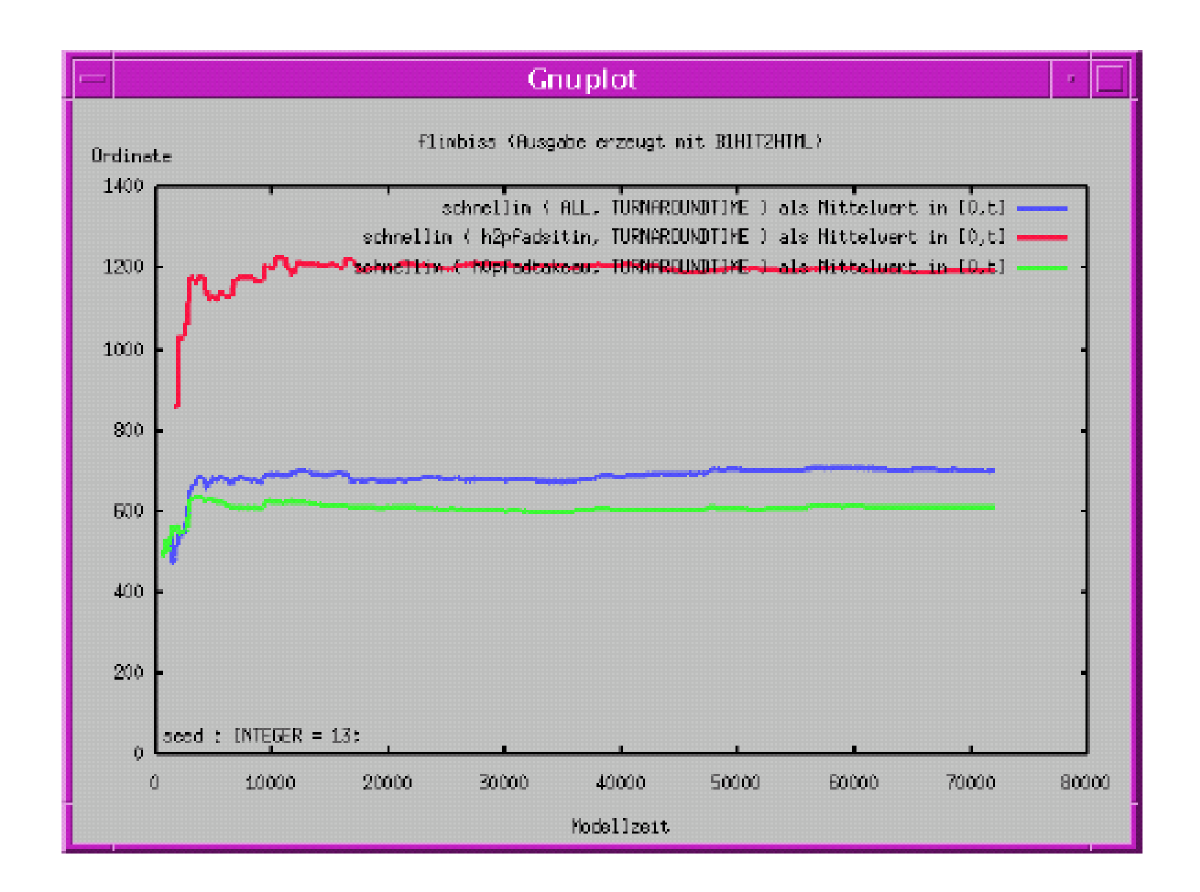


Abb. 8.13.: Visualisierung der Simulationsergebnisse

8.2. Modellierung mit stochastischen Petrinetzen und der APNN-Toolbox

Igor Gudovsikov

8.2.1. Einführung in Petrinetze

Petrinetze wurden 1962 von Carl Adam Petri erfunden und sind für die Beschreibung der Parallelität und Synchronisation in verschiedenen komplexen Systemen einsetzbar. Seitdem die Petrinetze zum ersten Mal von Petri beschrieben wurden, sind viele Veränderungen seiner ursprünglichen Netze definiert worden, wie z. B. farbige Petrinetze und eine Integration der Zeit in die stochastischen Petrinetze.

Petrinetze liefern eine bequeme graphische Darstellung des zu modellierenden Systems. Die Darstellung eines Petrinetzes enthält die folgenden Bestandteile:

- *Stellen*, gezeichnet durch Kreise. Das sind z. B. Zustände des Modells oder Objekte (bzw. Variablen) . Stellen können „Marken“ tragen.

- *Marken*, gezeichnet durch schwarze Punkte. Marken stellen den spezifischen Wert des Systemzustands (bzw. Wert einer Programmvariable) dar.
- *Transitionen*, gezeichnet durch Vierecke. Übergänge modellieren Prozesse, die die Werte von Bedingungen und von Zuständen ändern.
- *Kanten*, die Verbindungen der Stellen und der Transitionen spezifizieren, d. h. anzeigen, welche Gegenstände durch eine bestimmte Tätigkeit geändert werden.

Petrinetze sind bipartite Graphen. Daraus folgt, daß wir eine Stelle mit einer Transition oder umgekehrt verbinden dürfen, aber es ist verboten zwei Stellen oder Transitionen miteinander verbinden. Dieses würde auch nicht sinnvoll sein, wenn wir die Stellen und Transitionen in der Weise benutzen, wie wir diese definiert haben.

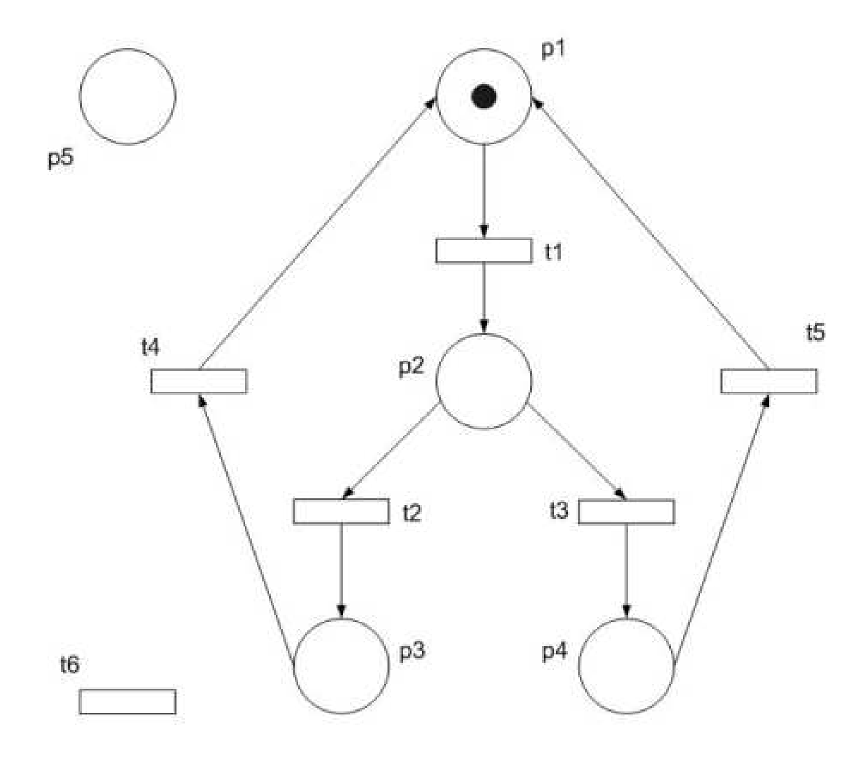


Abb. 8.14.: Beispiel eines Petrinetzes

In der Abbildung 8.14 ist die Stelle $p1$ an die Transition $t1$ angeschlossen. Stellen und Transitionen können mehrere eingehende und ausgehende Kanten haben. Die Stelle $p5$ und die Transition $t6$ sind an kein anderes Element angeschlossen. Solche Elemente werden *isolierte Stellen* oder *Transitionen* genannt. Isolierte Elemente eines Petrinetzes beeinflussen nicht den Rest des Netzes und folglich können wir sie vernachlässigen.

Bis jetzt haben wir nur die statische Struktur eines Petrinetzes beschrieben. Sein dynamisches Verhalten, kann über die folgenden Richtlinien ausgedrückt werden:

- *Aktivierung einer Transition*: Eine Transition wird als aktiviert bezeichnet, wenn alle ihre eingehenden Stellen mindestens eine Marke besitzen.
- *Feuern einer aktivierten Transition*: Eine aktivierte Transition kann feuern. Beim Feuern wird eine Marke auf jeder Inputstelle zerstört und eine Marke auf jeder ausgehende Stelle erstellt.

8.2.2. Einführung in die APNN-Toolbox

Die Entwicklung von diskreten, ereignisorientierten Systemen erfordert das Modellieren und die Analyse in allen Phasen des Systemdesigns, um Aussagen über Funktionalität und Leistung eines Systems schon während seiner Realisierung machen zu können. In den letzten Jahren sind Fortschritte in den Methoden und in den Techniken für Funktional- und Leistungsanalyse der komplizierten Software-Systeme gemacht worden. Aber um diese Techniken anwenden zu können, müssen sie in Software-Werkzeuge eingebunden werden, die auch modulare Spezifikation komplexer Systeme unterstützen. Hier werden wir eines dieser Tools, Abstract Petri Net Notation Toolbox (APNN), vorstellen. Die APNN-Toolbox wurde am Lehrstuhl IV der Universität Dortmund entwickelt und ist für die Modellierung und Analyse der dynamischen, ereignisdiskreten Systeme (Discrete event dynamic Systems, DEDS) gedacht.

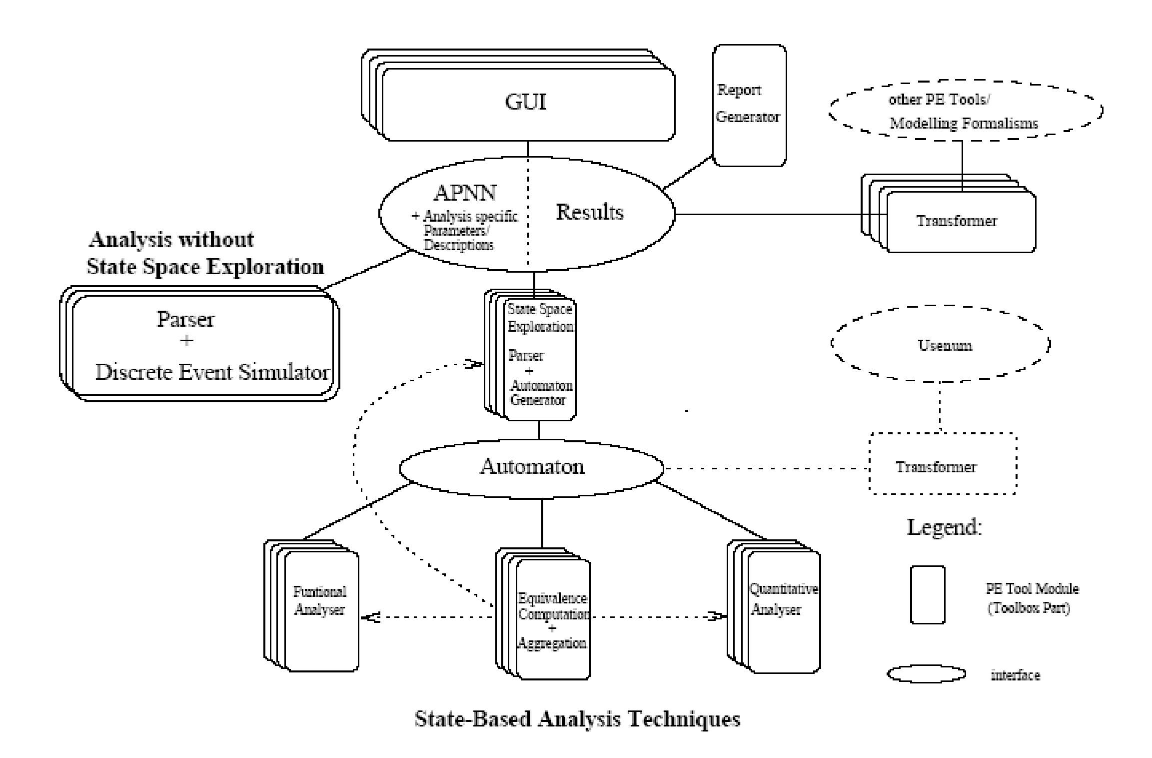


Abb. 8.15.: APNN-Toolbox Struktur

Abbildung 8.15 zeigt die allgemeine Struktur der APNN-Toolbox, welche aus mehreren Modulen besteht (Vierecke mit abgerundeten Ecken). Unterschiedliche Module tauschen Daten mit Hilfe von Datei-Interfaces (Ellipsen) aus, die kompatibel zu der APNN-Notation sind oder mit einer internen Notation für das Beschreiben von (stochastischen) Automaten übereinstimmen. Eine fettgedruckte Kante zwischen einem Interface und einem Modul bedeutet, daß dieses Modul in diesem Format lesen und schreiben wird. Normalerweise liest ein Modul die Beschreibung und schreibt das Resultat in demselben oder in einem anderen Format. Gepunktete Kanten mit Pfeilen beschreiben Abhängigkeiten zwischen Modulen.

8.2.3. Editor APNNed

Die APNN-Toolbox enthält eine graphische Benutzerschnittstelle (GUI), die in JAVA implementiert wurde. Der Editor APNNed (Abbildung 8.16) unterstützt die Spezifikationen der farbigen und stochastischen Petrinetze. APNNed enthält einige nützliche build-in Analysemöglichkeiten. Insbesondere wird das Markenspiel integriert. Die Animation des Spiels berücksichtigt die hierarchische Struktur des Netzes, indem es nur diejenigen Transitionen anzeigt, die sichtbar sind. Da das Markenspiel zu den so genannten *trace-driven*-Simulationen gehört, ist es möglich, das Token-Game rückwärts abzuspielen. Dies stellt eine wichtige Eigenschaft für das Debugging dar. Zu den anderen build-in's gehören z. B. die graphische Darstellung von Invarianten und der Verteilung der Marken auf Stellen des Netzes. In der internetfähigen Version kann das APNNed als Applet im Webbrowser verwendet werden. Das Applet stellt einige Funktionalitäten auf der Klientenseite zur Verfügung, während sich kompliziertere Analysemodule der APNN-Toolbox auf der Serverseite befinden. Diese Aufteilung erlaubt eine gemeinsame Arbeit in Gruppen.

APNN-Toolbox ist zu einigen anderen Modellierungswerkzeugen kompatibel, z. B. zum *ProC/B*-Tool zur Modellierung und Analyse von Prozessketten und logistischen Systemen. Technisch wird dies mit Hilfe von Schnittstellen bewerkstelligt, die die vorher erwähnte Notation auf die APNN-Notation abbilden.

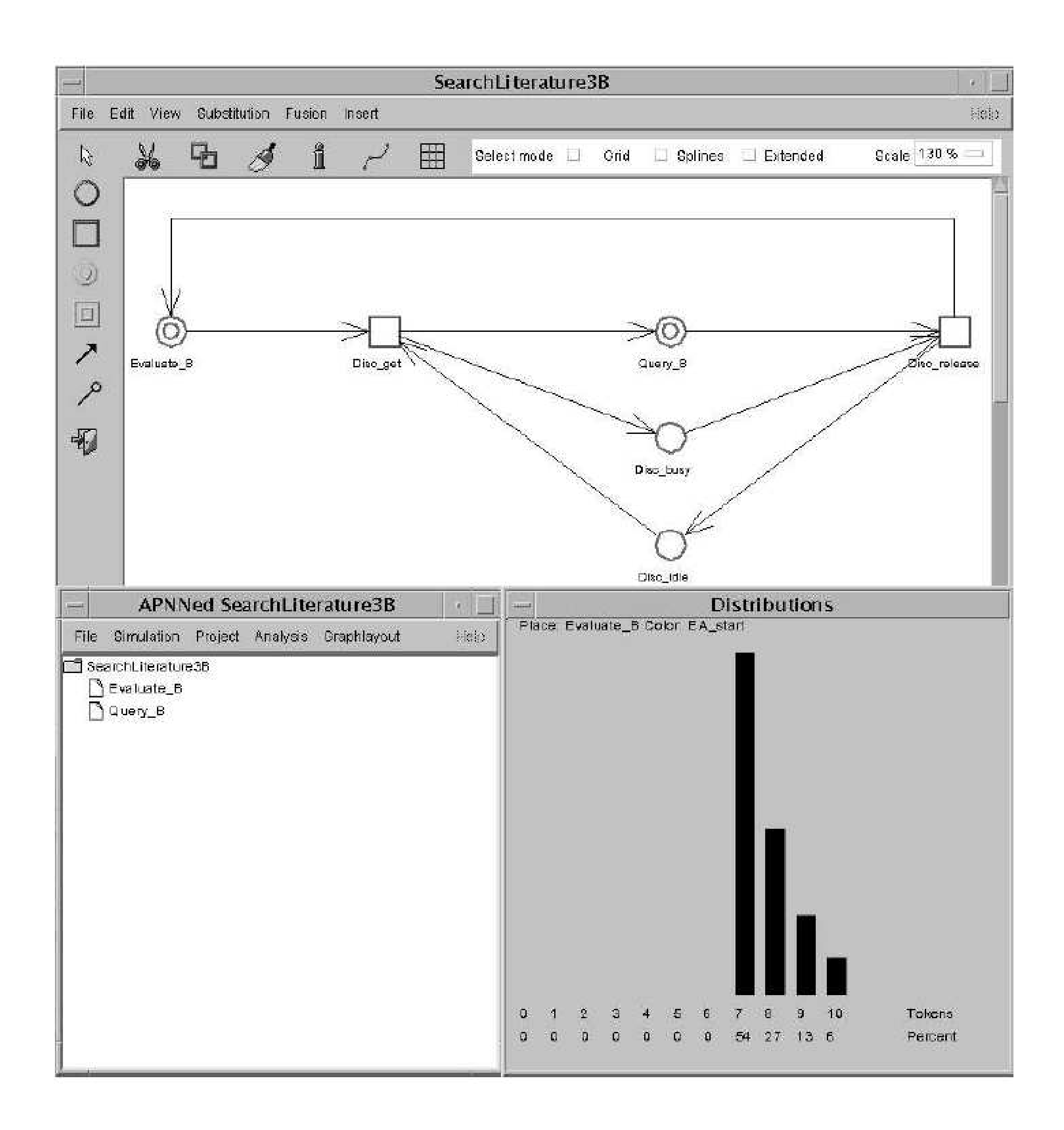


Abb. 8.16.: Der Editor APNNed

9. Simulationsmodelle

9.1. Modellierung und Analyse einer Stückgutumschlaghalle mit dem ProC/B Toolset

Michael Schaten, Jan Kriege

9.1.1. Modellentwurf

Durch die Modellierung und Analyse einer Stückgutumschlagshalle mit zwei Terminals T1 und T2 mit jeweils fünf Rampen sollte eine optimale Frequenz für die Wartung der in der Halle benutzten zwei Gabelstapler (es steht pro Terminal ein Gabelstapler zur Verfügung) gefunden werden, so daß der Durchsatz an den Terminals maximiert wird. Der Durchsatz der Halle bezeichnet die Menge der abgefertigten LKW, die die Terminals T1 und T2 anfahren. Sollte ein LKW die Halle zu einem Zeitpunkt erreichen, in dem alle 5 Rampen eines Terminals mit anderen LKW besetzt sind, so fährt dieser LKW unverrichteter Dinge wieder davon.
Wird das Wartungsintervall zu niedrig gesetzt, sinkt der Durchsatz, da während der Wartung keine LKWs abgefertigt werden. Allerdings können die Wartungsarbeiten an den Staplern auch nicht beliebig verzögert werden, da sie dann (möglicherweise sogar während der Abfertigung eines LKW) ausfielen. Ausfälle durch mechanische Fehlfunktionen oder ähnlichem treten im Mittel nach $30(+/-5)$ Tagen auf. Eine Reparatur eines ausgefallenen Staplers beansprucht sechs Tage, wo hingegen eine planmäßige Wartung lediglich einen Tag in Anspruch nimmt. Es ist daher sinnvoll, eine möglichst optimale Frequenz dieser Wartungsarbeiten herauszufinden.
Wesentlich in diesem Modell werden die stochastischen Schwankungen sein, die sich in vielerlei Hinsicht bemerkbar machen: So sind die Ankunftszeiten von zu be- oder entladenden LKWs exponentialverteilt mit einem Durchschnitt von 24 Minuten für Terminal 1 und 19 Minuten für Terminal 2. Der Ladevorgang bei diesen LKW ist ebenfalls exponentialverteilt und besitzt als Durchschnitt 36 Minuten zuzüglich 12 Minuten für das Reinigen des LKW und Erledigung der Papiere. Zu bemerken ist in diesem Zusammenhang, daß der Gabelstapler lediglich für den eigentlichen Ladevorgang benutzt wird und während der Bearbeitung der Papiere und Reinigung des LKW nicht genutzt wird.
Wie bereits im Absatz zu Beginn erwähnt, können diese Gabelstapler durch Fehlfunktionen außer Betrieb gesetzt werden. Dies geschieht zu normalverteilten Zeitpunkten mit einem Mittelwert von 30 Arbeitstagen. Für die daraufhin notwendige Reparatur des Staplers werden 6 Tage benötigt, während eine planmäßige Wartung lediglich 1 Tag in Anspruch nimmt. Auch diese beiden Werte sind Exponentiaverteilungen und stellen nur

einen Durchschnitt dar. Lediglich die Zeitspanne zwischen den Wartungen ist eine Konstante (25 Tage). Der Weg eines Lagerarbeiters von seinem Büro zu seinem Arbeitsgerät (Gabelstapler) ist dagegen wieder eine exponentialverteilte Zeitspanne (2, 4 Minuten).
Um das Modell und die vorkommenden Abläufe nicht unnötig zu verkomplizieren, wurde angenommen, daß die Arbeiter zwar nicht fest an einem Terminal beschäftigt sind, sich allerdings auch nicht gleichzeitig an einem Terminal aufhalten. Der Fall, daß einem Arbeiter während der 12-minütigen "clean-up-Phase" sein Gabelstapler von dem anderen Arbeiter "gestohlen" wird, ist damit ausgeschlossen. Dies vereinfacht das Modell stark, ohne dabei jedoch die Ergebnisse der Simulation zu verfälschen oder unbrauchbar zu machen.
Eine weitere vereinfachende Annahme ist die Wahl der Arbeitszeit: In diesem Modell wird 24 Stunden pro Tag gearbeitet, und auch Wochenenden bzw. freie Tage im Allgemeinen wurden nicht berücksichtigt.
Grafisch läßt sich das zu untersuchende Modell wie in Abbildung 9.1 veranschaulichen.

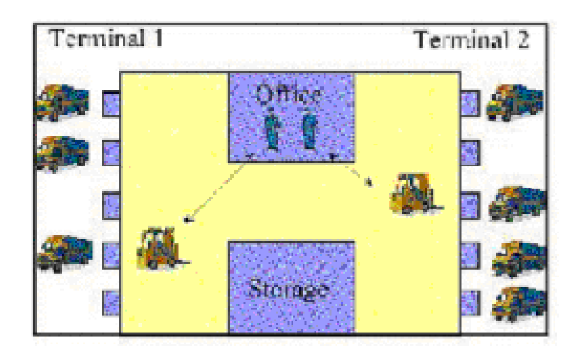


Abb. 9.1.: Stückgutumschlaghalle mit 2 Terminals und je 5 Rampen

Aus diesen bisherigen Betrachtungen ergeben sich also folgende Zustandsbeschreibenden Elemente:

- Position der beiden im Lager beschäftigten Mitarbeiter (Büro, Arbeitsweg oder einsatzbereit/beschäftigt auf einem Gabelstapler)
- Anzahl LKWs an den Rampen bzw. die Anzahl noch freier Rampen an einem der zwei Terminals → Anzahl LKWs pro Rampe muß für beide Terminals separat beschrieben werden.
- Zustand der Gabelstapler (beschädigt, in Wartung befindlich oder funktionstüchtig) → Zustand der Gabelstapler muß für beide Gabelstapler separat beschrieben werden.

Diese Elemente können durch folgende Ereignisse bzw. Zustandsverändernden Elemente beeinflußt werden:

- Ankunft von LKW an einem Terminal

- Beschädigung eines Gabelstaplers durch mechanische oder sonstige Probleme
- Wartung eines Gabelstaplers → Gabelstapler steht durchschnittlich einen Arbeitstag nicht zur Verfügung.
- Reparatur eines Gabelstaplers → Gabelstapler steht durchschnittlich sechs Arbeitstage nicht zur Verfügung.

9.1.2. Modellrealisierung

Die Stückgutumschlaghalle läßt sich im Wesentlichen durch vier Prozeßketten, drei Storages und zwei Prio-Servern als Prozeßkettenmodell darstellen.
Die beiden Terminals mit je fünf Rampen lassen sich gut durch ein Storage mit zweidimensionalem Vektor modellieren. Durch den angebotenen Dienst alter_or_skip kann das Verhalten der LKWs am Terminal (Entladen nur bei freier Rampe, sonst Abfahrt ohne Entladevorgang) einfach abgebildet werden. Die Arbeiter werden ebenfalls durch Storages modelliert.
Für die Modellierung der Gabelstapler eignen sich Prio-Server. Ein Gabelstapler kann durch insgesamt drei unterschiedliche Aktivitäten belegt werden: Entladung, Wartung und Reparatur. Reparaturen sollen dabei im Gegensatz zu Wartungsarbeiten Entladevorgänge unterbrechen. Durch Angabe geeigneter Prioritäten kann der Prio-Server dieses Verhalten darstellen.
Insgesamt vier Prozeßketten (zwei pro Terminal) beschreiben die zustandsverändernden Elemente des Modells. Eine Prozeßkette dient dabei jeweils zur Darstellung des Entladevorgangs, eine für Reparatur und Wartung. Im Folgenden die einzelnen Prozeßketten im Detail:

9.1.2.1. Entladevorgang

In exponentialverteilten Intervallen von durchschnittlich 24 bzw. 19 Minuten wird jeweils ein Prozeß erzeugt, der einen am Terminal ankommenden LKW darstellen soll. Über den Dienst alter_or_skip wird geprüft, ob am Terminal noch eine Laderampe frei ist. Falls nicht, endet der Prozeß sofort wieder; der LKW fährt also ohne Entladung wieder ab. Wenn eine Rampe frei ist, wird ein Arbeiter zum Entladen angefordert. Sobald der Arbeiter zur Verfügung steht, wird geprüft, ob der LKW immer noch entladen werden muß. Dies ist nötig, da ein Arbeiter alle am Terminal stehenden LKWs entlädt bevor er zum Office zurückkehrt. Es kann also sein, daß der Arbeiter zum Zeitpunkt zu dem er angefordert wurde, bereits auf dem Weg zum Terminal war oder dort bereits andere LKWs entladen hat. Der LKW, der den Arbeiter gerade angefordert hat, kann also von dem bereits am Terminal anwesenden Arbeiter entladen werden und die Anforderung ist danach nicht mehr nötig. Sobald ein Arbeiter zur Verfügung steht und ein LKW entladen werden muß, macht sich der Arbeiter auf den Weg zum Terminal. In einer Schleife wird er dort solange LKWs entladen bis alle vorhandenen LKWs abgearbeitet sind.
Der Arbeitsvorgang besteht aus einer Entladephase für die der Gabelstapler angefordert

wird (exponentialverteilt mit Mittelwert 36 Minuten) und einer Aufräumphase (exponentialverteilt mit Mittelwert 12 Minuten). Sind alle LKWs entladen, begibt sich der Arbeiter zurück in sein Büro.
Abbildungen 9.2, 9.3 und 9.4 zeigen den Entladevorgang als Prozeßkette in ProC/B.

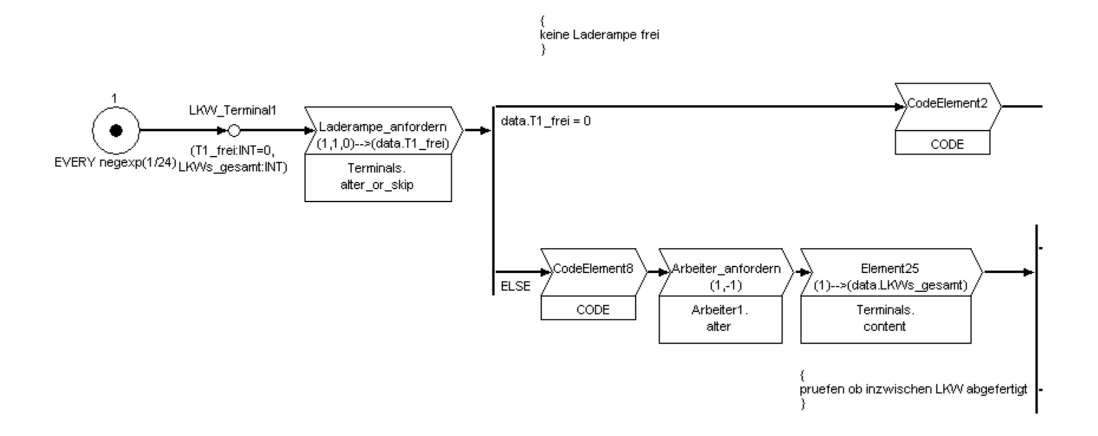


Abb. 9.2.: Entladevorgang Teil 1

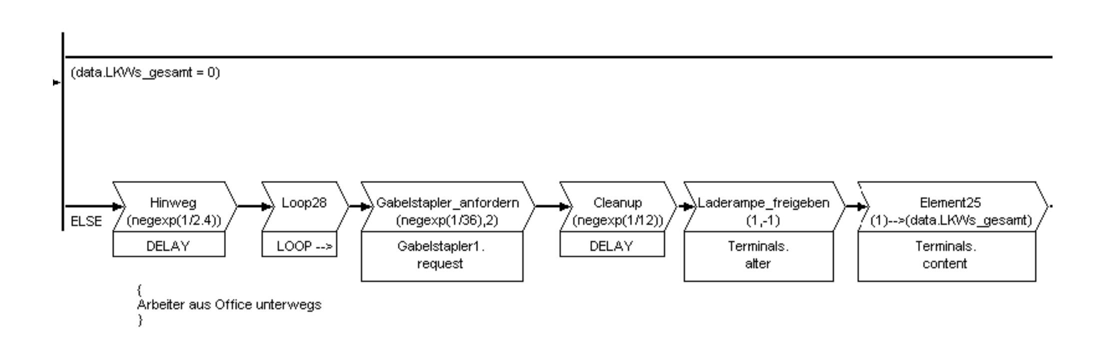


Abb. 9.3.: Entladevorgang Teil 2

9.1.2.2. Reparatur/Wartung

Die Reparatur- und Wartungsvorgänge sind pro Gabelstapler in einer Prozeßkette dargestellt, da sich die Reparatur- und Wartungszeitpunkte gegenseitig beeinflussen: Z.B.

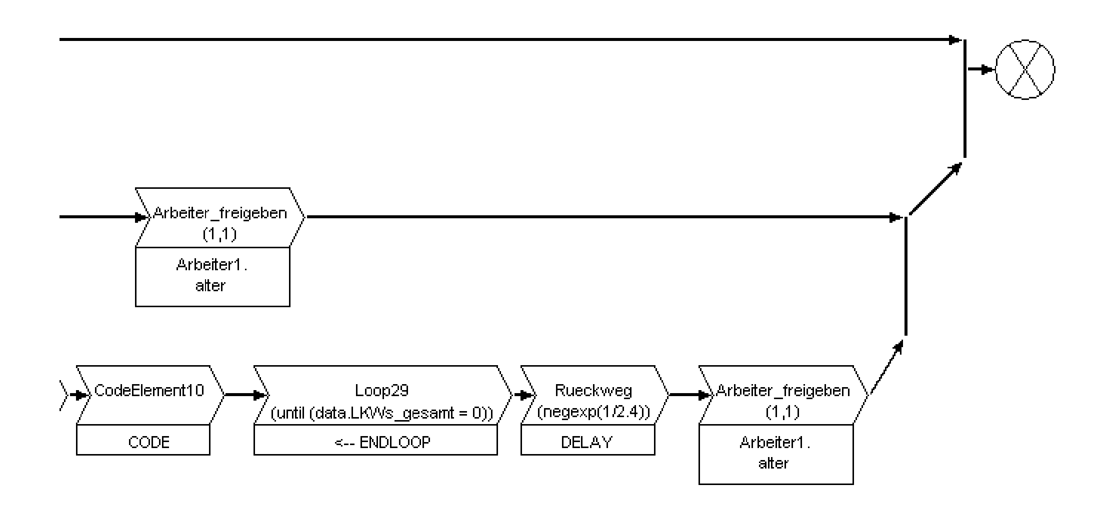


Abb. 9.4.: Entladevorgang Teil 3

nach erfolgter Wartung muß die Zeit zum nächsten Ausfall zurückgesetzt werden.
Die Ausfallzeitpunkte durch Reparatur sind normalverteilt mit Mittelwert 40 Tage und einer Standardabweichung von 5 Tagen, die Wartungsintervalle haben einen festen Wert von 25 Tagen. Hierfür soll später das optimale Intervall gefunden werden.
Zum Zeitpunkt 0 wird der Prozeß für Wartung und Reparatur gestartet. Es wird der Zeitpunkt der nächsten Reparatur festgelegt und danach durchläuft der Prozeß eine Endlosschleife. Jede Minute wird geprüft, ob ein Wartungs- oder Reparaturzeitpunkt erreicht wurde. Für Reparatur und Wartung wird jeweils der Gabelstapler für 6 Tage oder 1 Tag (jeweils Mittel einer Exponentialverteilung) angefordert. Danach wird der Zeitpunkt des nächsten Ausfalls bestimmt.
Abbildung 9.5 zeigt den Reparatur- und Wartungsvorgang als Prozeßkette in ProC/B.

9.1.3. Simulation des Modells und Ergebnisse

Das Modell der Stückgutumschlaghalle wurde mit verschiedenen Wartungsintervallen der beiden Gabelstapler in einem Zeitraum von 23 bis 30 Tagen untersucht. Dabei wurden die Wartungsintervalle schrittweise um je 12 Stunden erhöht.
Der maximal mögliche Durchsatz bei durchschnittlichen Ankunftszeiten der LKWs von 19 bzw. 24 Minuten liegt bei etwa $0,094$ LKWs pro Minute. Die erwarteten Durchsätze liegen aufgrund der Entladezeiten für die LKWs und der Ausfälle durch Wartung und Reparatur der Gabelstapler natürlich deutlich niedriger.
Unsere Simulationsergebnisse (siehe Abbildung 9.6) wiesen relativ starke Schwanken auf. Der im untersuchten Bereich beste Wert liegt bei 36000 Minuten (25 Tage) als Wartungsintervall für Gabelstapler 1 und 38160 Minuten ($26,5$ Tage) für Gabelstapler 2. Hier wurde ein Durchsatz von $0,042$ erzielt, was insgesamt 5983 abgefertigten

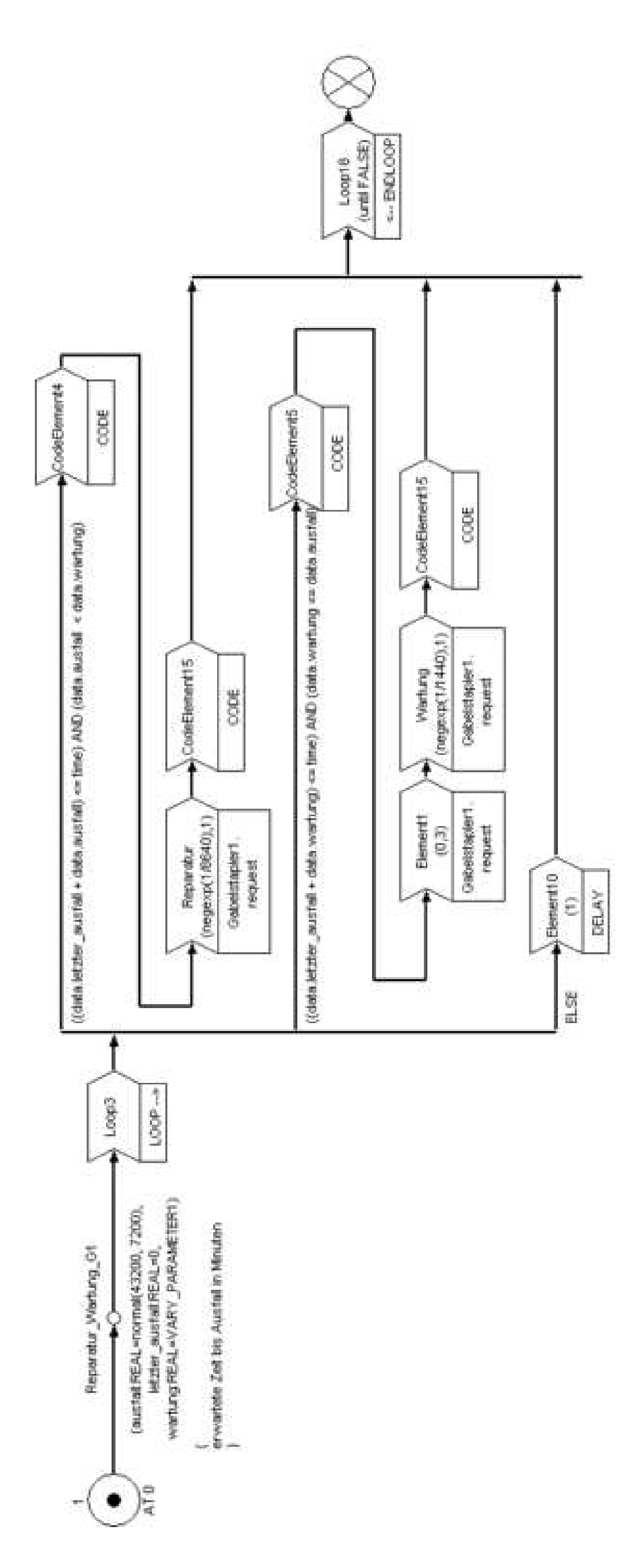


Abb. 9.5.: Reparatur und Wartung

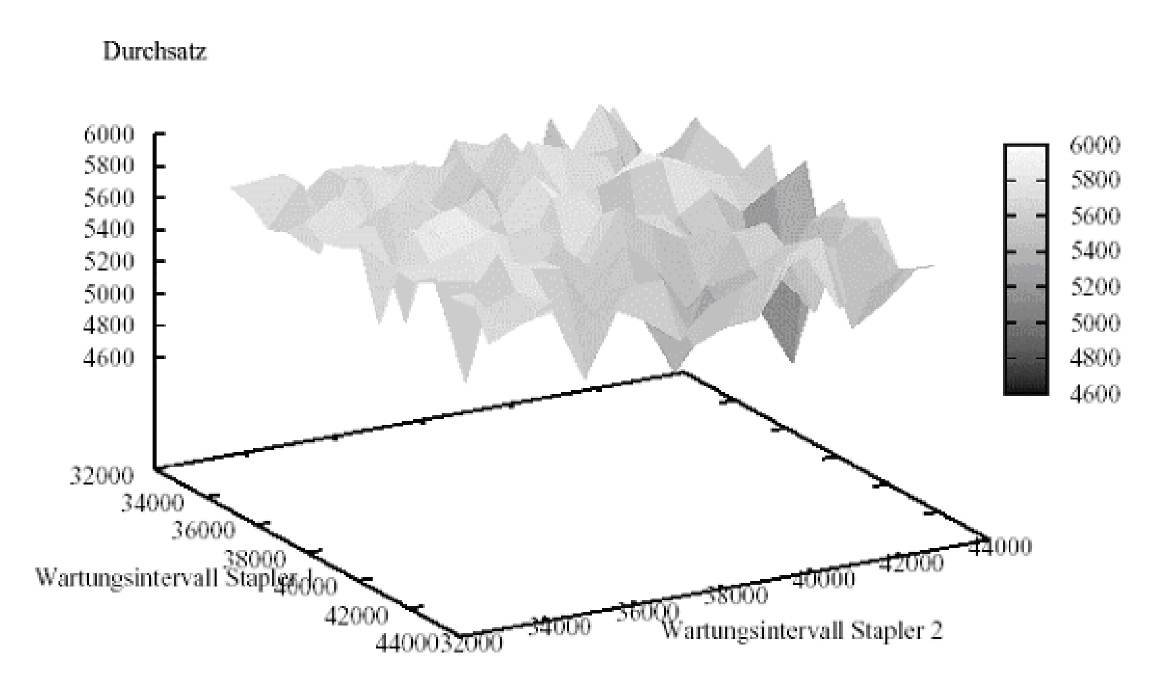


Abb. 9.6.: Simulationsergebnisse

LKWs entspricht. Insgesamt findet man in diesem Bereich eine relative Häufung von hohen Durchsatzwerten, so daß wir in der Nähe dieses Punktes auch das Optimum des untersuchten Intervalls vermuten.

9.2. Modellierung und Analyse einer Stückgutumschlagshalle mit der APNN-Toolbox

Normann Powierski, (Christian Bäcker)

9.2.1. Anforderungen

Das Ziel der Modellierung ist eine präzise Umsetzung der gegebenen Situation, mit der Möglichkeit durch Variation der Variablen eine Maximierung des Umschlages bei möglichst langen Wartungsintervallen zu ermöglichen.
Wesentliche Aspekte die zu beachten sind werden im Folgenden aufgelistet:

- 2 Terminals mit je 5 Rampen, je einem Gabelstapler sowie 2 Arbeitern
- LKW kommen an Terminal 1 mit einer Exp. Verteilung mit Mittelwert 24 Min. an.

- LKW kommen an Terminal 2 mit einer Exp. Verteilung mit Mittelwert 19 Min. an.
- LKW fahren an eine freie Rampe ihres Terminals. Falls keine Rampe frei ist, fahren sie wieder ab.
- Bei Ankunft eines LKW an einer Rampe kommt ein Arbeiter vom Büro zur Rampe sofern er nicht schon da ist.
- Der Weg eines Arbeiters vom Büro zur Rampe oder zurück dauert 2,4 Min.
- Ein Entladevorgang mit dem Gabelstapler dauert durchschnittlich 36 Min.
- Das Aufräumen nach dem Entladen dauert 12 Minuten.
- Die Gabelstapler werden nach 25 Tagen gewartet.
- Die Gefahr eines Ausfalls ohne Wartung beträgt normalverteilt 30 Tage mit einer Standardabweichung von 5 Tagen.
- Die Dauer einer Reparatur ist exponentiell verteilt mit dem Mittelwert 6 Tage.
- Die Dauer einer Wartung ist exponentiell verteilt mit dem Mittelwert 1 Tag.

9.2.1.1. Randbedingungen

- Falls nach einem Entladevorgang bereits ein weiterer LKW an einer anderen Rampe desselben Terminals steht, fängt der Arbeiter an, dort zu entladen. Ansonsten geht er zurück zum Büro (Dauer entsprechend 2,4 Min).
- Wird während eines Entladevorganges ein Wartungsintervall fällig, wird der LKW noch vollständig entladen und erst dann wird der Stapler gewartet.
- Bei einem Ausfall des Gapelstaplers während eines Entladevorganges, wird der Entladevorgang für die Dauer der Reparatur unterbrochen.

9.2.1.2. Modellanahmen

- Es gibt unendlich viele LKW die irgendwann mal am Terminal ankommen können.
- Nicht abgefertigte LKW verursachen in diesem Modell keine Warteschlange und werden somit ignoriert. Es werden keine Informationen über abgewiesene LKW geführt.
- Entladene LKW fahren einfach weg und es werden keine weiteren Annahmen mehr darüber gemacht.
- Das Lager ist unendlich groß und wird niemals voll.
- Die Arbeiter machen keine Pausen und Entladen entweder oder warten auf Arbeit.

9.2.1.3. Zustandsverändernde Elemente im Modell

- Arbeiter: Ist entweder im Büro oder am Terminal.
- Stapler: Funktioniert oder ist defekt, bzw. wird gewartet oder repariert.
- Rampen: Sind besetzt oder frei.

9.2.1.4. Zustandsbeschreibende Elemente im Modell

- Es gibt 1 Lager
- Es gibt 2 Terminals
- An den Terminals sind jeweils 5 Rampen
- Es gibt 2 Arbeiter
- Pro Terminal gibt es einen Stapler

9.2.2. Umsetzung

Die Umsetzung der Stückgutumschlaghalle als APNN-Modell ist in Abbildung 9.7 zu sehen. Die beiden Terminals des Modells sind prinzipiell unabhängig voneinander, da jeder Arbeiter an seinem Terminal arbeitet und wartet sofern der Stapler gewartet oder repariert wird.

Zur Darstellung des Wartungs- bzw. Reparaturvorgangs wurde ein eigenständigen Prozess modelliert, der den Stapler entweder nach der gegebenen Zeit wartet, oder im Falle einer Reparatur, den Stapler aus dem aktuellen Entladeprozess herausnimmt. Zum Zählen der Wartungs- bzw. Reparaturintervalle befinden sich am Ende des Modells noch ,,Endstellen“ in der die Token gesammelt werden. Zum Zählen des Gesamtumschlages befindet sich ebenfalls eine ,,Endstelle“ an der rechten Seite des Modells.

9.2.3. Simulation

Zur Erstellung des Response Surface, wurden zwei Simulationen durchgeführt:

Simulationslauf 1 *Laufzeit: 5 Jahre*
Wartungsintervall: jeweils von 10-40 Tagen
Die grafische Darstellung der Ergebnisse ist in Abbildung 9.8 zu sehen.
Die besten Werte für den LKW Durchsatz erhält man bei Wartungsintervallen um 20 Tage für beide Stapler.

Simulationslauf 2 *Laufzeit: 10 Jahre*
Wartungsintervall: jeweils von 21-25 Tagen
Wie man in Abbildung 9.9 allerdings erkennen kann, liegt ein Optimum auch mit längerer Simulationsdauer bei dem Wert 20 / 20. Die kürzere Laufzeit der vorherigen Simulation führte also nicht zu verfälschten Werten.

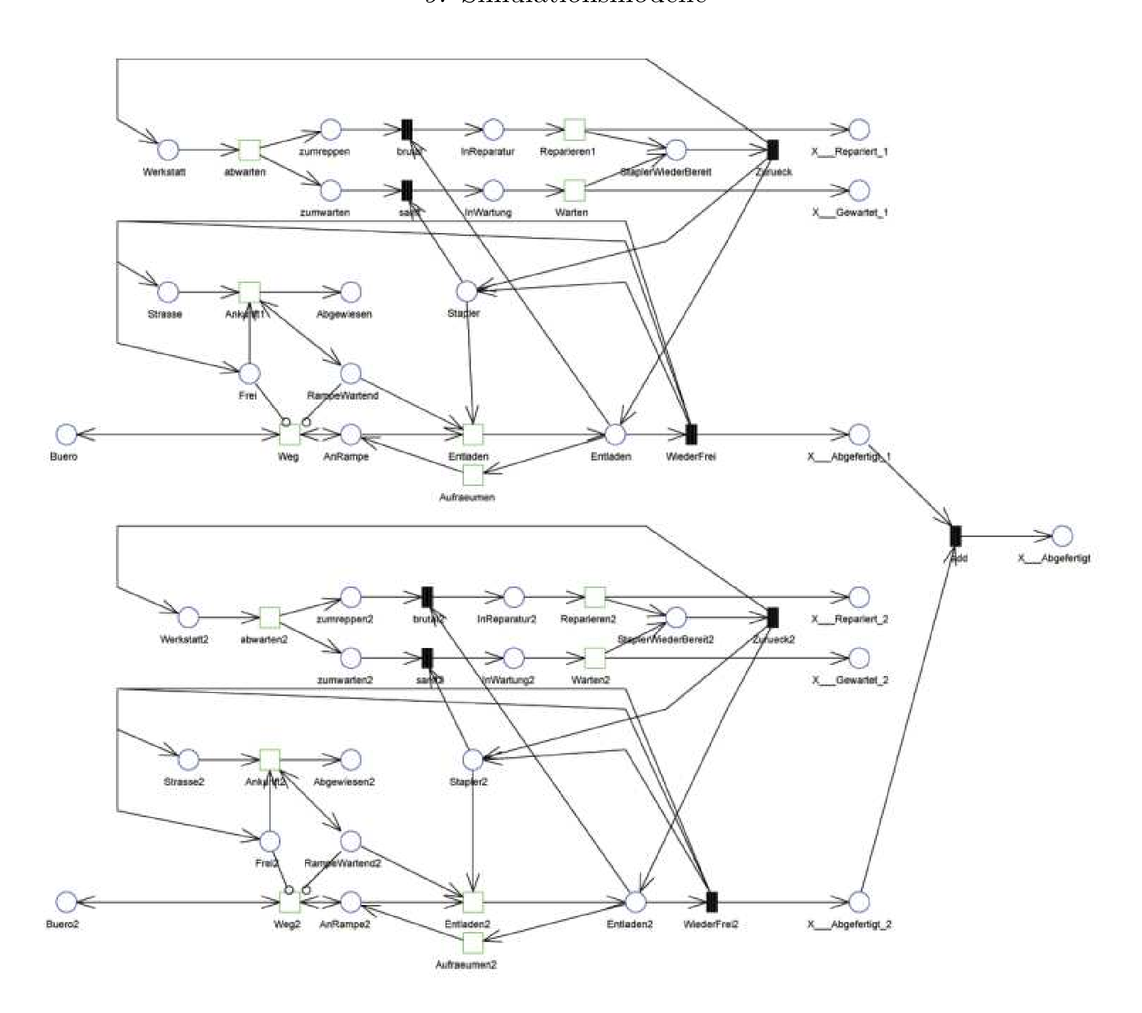


Abb. 9.7.: Modellskizze der Stückgutumschlagshalle mit der APNN-Toolbox

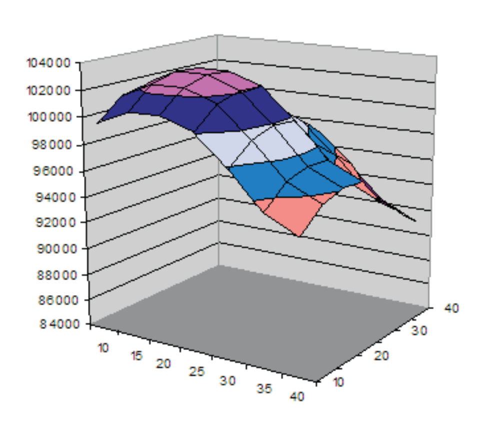


Abb. 9.8.: APNN Modell Stückgutumschlagshalle: Response Surface 5 Jahre Laufzeit

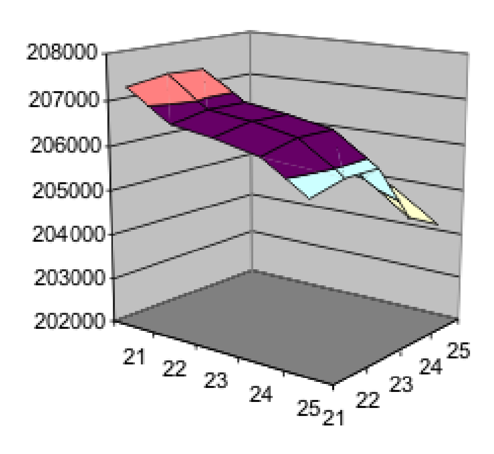


Abb. 9.9.: APNN Modell Stückgutumschlagshalle: Response Surface 10 Jahre Laufzeit

9.3. Modellierung und Analyse einer Ampelkreuzung mit der APNN-Toolbox

Katharina Balzer, Julia Hielscher

9.3.1. Modellbeschreibung

Es soll die Kreuzung einer großen und einer kleineren Straße als Petrinetz modelliert werden, wobei der Verkehrsfluß durch Ampeln geregelt werden soll. Desweiteren können Fußgänger jede der beiden Straßen überqueren. Ein Freiheitsgrad ist die Wahl der Ampeln bzw. die Entscheidung über zusätzliche Linksabbieger- oder Rechtsabbiegerpfeile. Es liegen Angaben vor über die Zeit, die Autos und Fußgänger mindestens brauchen, um die Kreuzung zu überqueren. Außerdem ist bekannt, wie viele Autos von welcher Straße aus nach links und nach rechts abbiegen. Dazu ist noch das Verkehrsaufkommen zu bestimmten Tageszeiten aufgelistet.

Das Ziel unserer Modellierung ist es, den Verkehrsteilnehmern ein sicheres und möglichst zügiges Vorankommen zu ermöglichen. Die Ampeln sollen also einerseits so geschaltet sein, daß bei regelgerechtem Verhalten der Autofahrer und Fußgänger keine Unfälle geschehen und andererseits aber auch zu lange Warteschlangen vermieden werden. Die Länge der Warteschlangen vor den Ampeln ist also zu minimieren. Dazu werden die Grünphasen der Ampeln variiert. Dabei ist zu beachten, daß der Aufbau der Kreuzung (die Anzahl der Fahrspuren, Links-/Rechtsabbiegerpfeile, etc.) den Verkehrsfluß beeinträchtigt.
Der Zustand der Kreuzung wird beschrieben durch die Zustände der Ampeln, also durch die Farben, die gerade aufleuchten. Verändert werden die Zustände über die Zeit. Im weiteren untersuchen wir das folgende Szenario: An jeder Ecke der Kreuzung gibt es nur eine Ampel und eine Fahrspur. Ist die Ampel grün, so darf sowohl nach rechts, nach links als auch geradeaus gefahren werden. Das gleiche gilt für die Fußgängerampeln. Sie werden ebenfalls parallel und zusätzlich auch parallel zu den jeweiligen Autoampeln geschaltet. Uns reicht zur Modellierung allerdings eine Ampel, da gegenüberliegende Ampeln ohnehin das gleiche anzeigen, und die Verkehrsteilnehmer der einen Richtung die Straße überqueren dürfen, wenn in der anderen Richtung Rot angezeigt wird. D. h. die Rotphase der einen Richtung entspricht der Grünphase der anderen Richtung.

Die wesentlichen Aspekte in unserem Modell sind zum einen die Anzahl der Ampeln, und zum anderen die Tatsache, daß Linksabbieger sowohl den Gegenverkehr als auch die Fußgänger beachten und daß Rechtsabbieger ebenfalls auf die die Straße überquerenden Fußgänger achten müssen. Abstrahiert haben wir in unserem Modell von den Zeiten, die die jeweiligen Verkehrsteilnehmer je nach Situation zum Überqueren der Kreuzung benötigen.

9.3.2. Umsetzung in ein APNN-Modell

Bei der Umsetzung des Modells der Ampelschaltung in ein generalisiertes stochastisches Petrinetz haben wir die folgenden Akteure mit den folgenden Aktionen unterschieden:

die Ampel: Sie ändert ihren Zustand über die Zeit, wechselt also von Grün nach Gelb nach Rot, usw.

die Autos: Sie kommen an der Kreuzung an, warten eventuell vor der Ampel, fahren dann auf die Kreuzung (links, geradeaus oder rechts) und verlassen nach einem optionalen Wartevorgang die Kreuzung wieder.

die Fußgänger: Sie kommen an einer Fußgängerampel an, warten, wenn Rot ist und überqueren die Straße, wenn die Ampel Grün wird. Fußgänger haben immer Vorrang vor allen anderen Verkehrsteilnehmern, in diesem Fall den Autos.

Die Ampel hat in unserem Petrinetzmodell die in Abbildung 9.10 dargestellte Struktur.

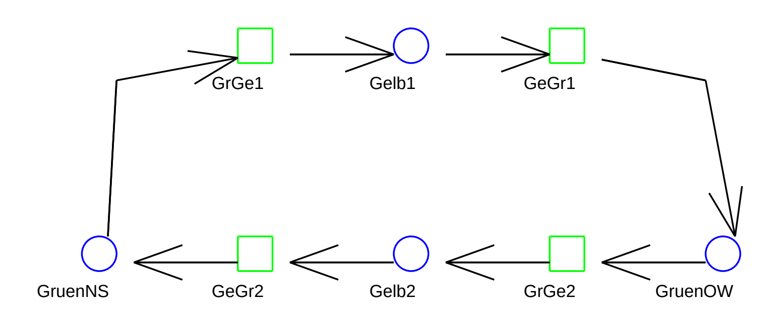


Abb. 9.10.: Eine Ampel als Petrinetz

Im Szenario haben wir Stellen jeweils für die Zustände Grün und Gelb der Ampel. Es gibt eine Grünphase für die Autos, die sich auf Straße 1 befinden und eine für die Autos auf Straße 2. Die zweite Gelbphase liegt darin begründet, daß die Grünphase der einen Richtung, die Rotphase der anderen Richtung darstellt, und auch diese Richtung eine Gelbphase haben muß, damit die auf der Kreuzung wartenden Autos diese auch nach dem Umschalten der Ampel noch verlassen können. Die Transitionen zwischen den einzelnen Stellen sind zeitbehaftet und zwar mit einer konstanten Verteilung. Dabei dauert jede Gelbphase genau fünf Sekunden. Die Grünphasen variieren. Sie sind zu optimieren, so daß möglichst viele Autos die Kreuzung überqueren können.

In Abbildung 9.11 ist eine Fahrtrichtung einer Straße, zum Beispiel Nord nach Süd, dargestellt. Wir haben zum einen je eine Stelle, die die Warteschlange der Autos vor der Ampel und ihr Verschwinden hinter der Kreuzung darstellt. Auf der Kreuzung haben wir drei Stellen angeordnet, eine für die Linksabbieger, die eventuell warten müssen, eine für die Rechtsabbieger und eine für diejenigen Autos, die geradeaus fahren wollen. Diese Unterscheidung ist notwendig, da bei Linksabbiegern z. B. überprüft werden muß, ob kein Gegenverkehr kommt und ob niemand in dieselbe Straße rechtsabbiegen will. Die Transitionen von der Stelle vor der Kreuzung zu den Stellen auf der Kreuzung haben wir zeitlos gewählt, da bestimmte Prozentzahlen abbiegen und geradeaus fahren sollen. Diese werden als Gewichte an die zeitlosen Transitionen geschrieben. Die Transition ganz am

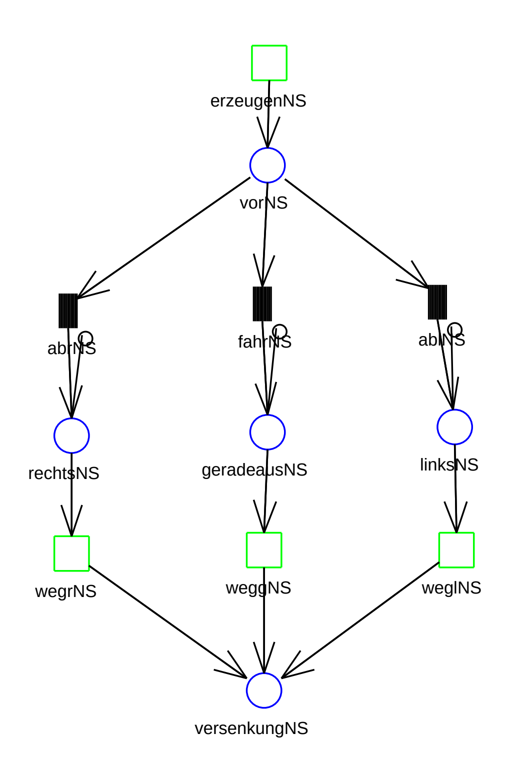


Abb. 9.11.: Fahrtrichtung Nord–Süd einer Straße als Petrinetz

Anfang, die keine Input-Stellen besitzt, erzeugt Autos gemäß einer Exponentialverteilung mit einer variierenden Rate. Das Gleiche gilt für die Transitionen ohne Input-Stellen im Modell des Fußgängers.

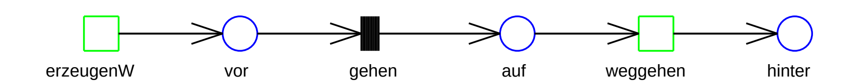


Abb. 9.12.: Modellierung eines Fußgängers als Petrinetz

Abbildung 9.12 zeigt die Modellierung eines Fußgängers als Petrinetz. Wir haben drei Stellen vorgesehen, je eine für vor der Straße, auf der Straße und hinter der Straße. Das Betreten der Straße haben wir als zeitlose Transition modelliert, da Fußgänger erst einmal sofort die Straße betreten, sobald sie Grün haben. Danach verlassen sie diese exponentialverteilt, da ja nicht jeder Fußgänger gleich schnell über die Straße geht.

Um die verschiedenen Fahrtrichtungen der Straßen, die Fußgänger und die Ampel zusammenzubringen, haben wir mit Inhibitor-Kanten und Überprüfungen in Form von

Pfeilen in beide Richtungen gearbeitet (siehe Abbildung 9.13).

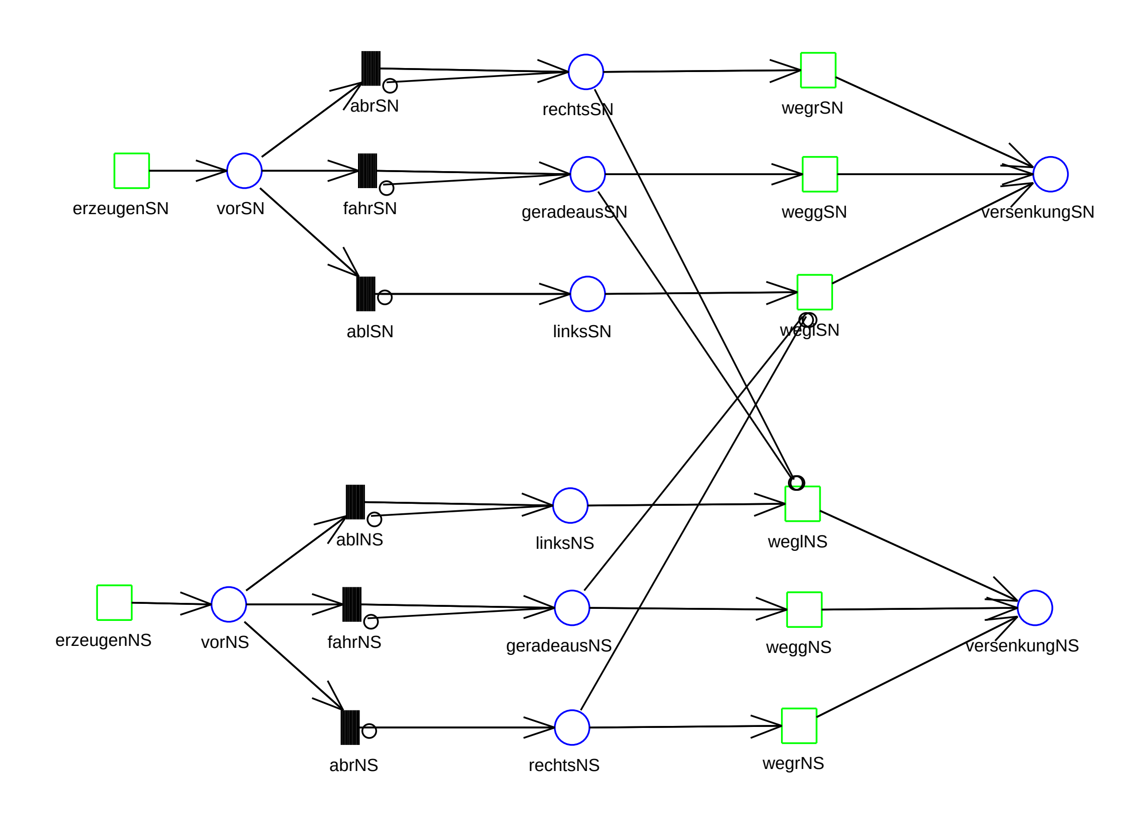


Abb. 9.13.: Zusammenhänge zwischen zwei Fahrtrichtungen einer Straße

Überprüft werden muß vor allem, wann die Ampel „GruenNS“ ist. Erst wenn dies der Fall ist, können Autos und Fußgänger in Richtung Nord–Süd und Süd–Nord die Kreuzung überqueren. Wenn die Ampel „GruenOW“ zeigt, so ist dies die Grünphase für die Richtungen Ost–West und West–Ost. Also muß auch dies überprüft werden. Alle anderen Überprüfungen finden mithilfe von Inhibitor-Kanten statt. Zum Beispiel zeigt eine Inhibitor-Kante von der Stelle “auf der Kreuzung“ eines Fußgängers auf die Transition des Rechtsabbiegers derjenigen Straße, die dorthin rechtsabbiegen würde, wo der Fußgänger gerade die Straße überquert. Mit der Inhibitor-Kante wird nun erreicht, daß das rechtsabbiegende Auto auf der Kreuzung warten muß, bis alle Fußgänger die Straße verlassen haben. Das gleiche gilt für Linksabbieger und die ihnen entgegen kommenden geradeaus fahrenden Autos. So wird alles überprüft, was zu einem Unfall führen könnte. Was außerdem mittels einer Inhibitor-Kante überprüft wird ist, wie viele Autos sich gleichzeitig auf der Kreuzung befinden. Dies dürfen jeweils für links, rechts und geradeaus maximal drei sein. Die gesamte Kreuzung als Petrinetz ist in Abbildung 9.14 zu sehen.

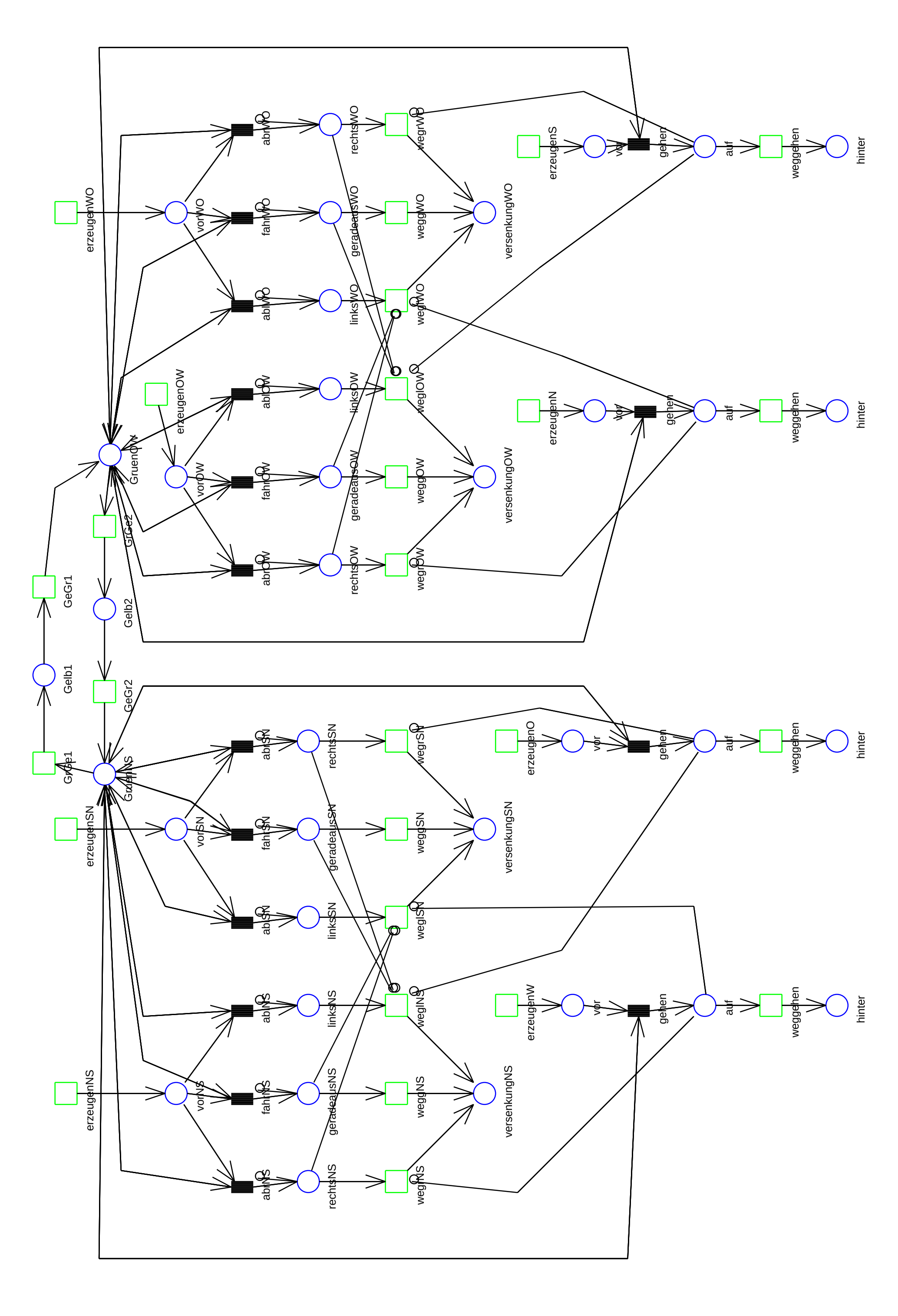


Abb. 9.14.: Das Modell der Ampelkreuzung

9.4. Modellierung und Analyse eines Fertigungssystems mit der APNN-Toolbox

Thomas Hutter, Marcus van Elst

9.4.1. Modellentwurf

In dem Beispiel geht es um eine Produktionskette bestehend aus vier Produktionsstationen. Jede dieser Produktionsstationen kann aus einer oder mehreren Maschinen bestehen, wobei mehrere Maschinen den Arbeitsablauf an dieser Station beschleunigen, weil der Produktionsdurchsatz gesteigert wird.

Die erste Produktionsstation bekommt die zu bearbeitenden Rohstoffe aus einem theoretisch unendlich großen Lager, bearbeitet diese und legt die bearbeiteten Teile in einen Zwischenpuffer. Sollte dieser Zwischenpuffer bereits voll sein, so pausiert diese Arbeitsstation so lange, bis wieder Platz frei ist. Eine pausierende Produktionsstation beginnt sogleich wieder mit der Arbeit, sobald der versorgende Zwischenpuffer wieder mit mindestens einem Teil gefüllt ist und ihr Zielpuffer ausreichende Kapazität hat. Die vierte, letzte, Produktionsstation stellt das Gesamtprodukt fertig und besitzt keinen Zielpuffer.

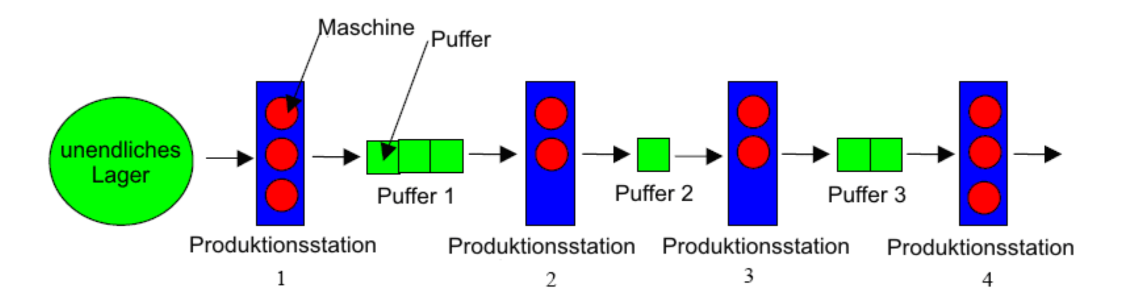


Abb. 9.15.: Produktionsablauf

Die Arbeitszeiten je Maschine sind exponentiell verteilt, die Mittelwerte ergeben sich aus Tabelle 9.1.

Produktionsstation	Mittlere Arbeitszeit je Maschine in Stunden
1	$\frac{1}{3}$
2	$\frac{1}{2}$
3	$\frac{1}{5}$
4	$\frac{1}{4}$

Tabelle 9.1.: Die Verteilung der einzelnen Maschinen

An jeder Station können unabhängig von den anderen Stationen zwischen 1 und 3

Maschinen eingesetzt werden, die drei Zielpuffer hinter den Stationen 1, 2 und 3 können unterschiedliche Kapazitäten zwischen 1 und 10 haben.

Ziel der Simulation ist es, verschiedene Kombinationen aus Maschinen pro Station und Puffergrößen experimentell auszuprobieren und auszuwerten.

9.4.2. Modellrealisierung

Jede Produktionsstation ist durch eine Menge von Aktionen und Bedingungen dargestellt, jeder der drei Zielpuffer jeweils durch eine einzelne Bedingung. Vereinfacht betrachtet wird eine schwarze Rohstoffmarke vom Lager durch die einzelnen Produktionsstationen und Puffer bis zum Ende weitergereicht.

Die Realisierung der unterschiedlich großen Puffer ist mit Hilfe farbiger Marken entstanden, die jeweils die Kapazität der einzelnen Puffer repräsentieren.

Jede Produktionsstation beginnt mit drei zeitlosen Transaktionen für maximal drei eingesetzte Maschinen. Mit Hilfe eines Kreislaufs, in dem spezielle Maschinenmarken zwischen jeweils einer zeitlosen Transaktion, einem Zwischenpuffer nur für die jeweilige Maschine und der eigentlichen verzögernden Maschine rotieren, wird jede der maximal drei Maschinen realisiert.

Die zeitlosen Transaktionen können nur dann feuern, wenn aus dem Lager eine schwarze Marke sowie aus dem Zielpuffer eine farbige Marke und von Back eine Maschinen Marke bereitsteht. Folglich verringert sich die Menge an Marken im Lager, im Zielpuffer und im Back Ereignis. Die schwarze Marke sowie die Maschinenmarke werden in den Zwischenpuffer "weitergereicht", von wo aus sich die eigentliche Maschine "bedient" und eine Zeitverzögerung auslöst. Anschließend wird die Maschinenmarke zurück zur zeitlosen Transaktion geführt (Kreislauf), die schwarze Rohstoffmarke wird in den Zielpuffer überführt. Dieses Konzept wird bis Produktionsstation 4 entsprechend fortgeführt. Die Summe aus schwarzen und farbigen Marken je Puffer ergibt also stets die gesamte Pufferkapazität im vorliegenden Puffer.

Abbildung 9.16 stellt genau eine Workstation dar, also ein Viertel des Gesamtmodells. Links befindet sich das Lager (bzw. bei den folgenden Workstations der Versorgungs- oder Zwischenpuffer), in der Mitte sind untereinander drei Maschinen mit ihrem Zwischenlager für die technische Realisierung, rechts befindet sich der nächstfolgende Zwischenpuffer bzw. ganz am Ende das Endlager, welches die fertiggestellten Teile für die Auswertung der Gewinnfunktion zählt. Das gesamte Modell hat den in Abb. 9.17 dargestellten Aufbau.

9.4.3. Validierung

Die einzelnen Workstations werden mit unterschiedlich vielen Maschinen bestückt, die nachfolgenden Puffer sind unterschiedlich groß, die Modellzeit beträgt 720 Stunden bei 1.000.000 Replikationen.

Mathematisch betrachtet lautet die Gewinnfunktion g also bei n Maschinen, p Pufferkapazität und t fertiggestellten Teilen:

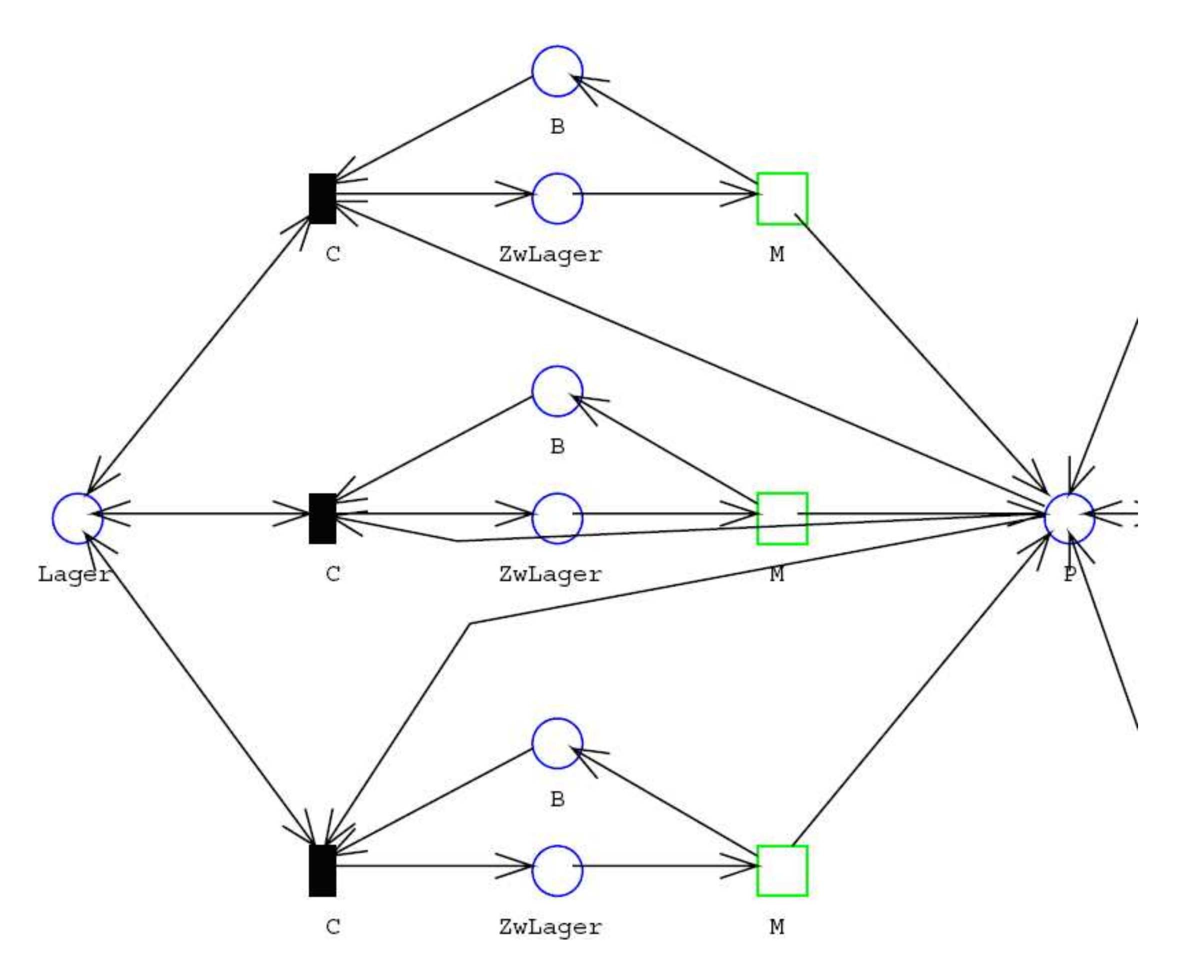


Abb. 9.16.: Kreislauf einer Workstation

$$g(n, p, t) = 200 \cdot t - 25000 \cdot n - 1000 \cdot p$$

9.4.4. Experimentieren

20 Systemkonfigurationen sollen uns Aufschluß über Funktions- und Leistungsfähigkeit des in der APNN-Toolbox modellierten Modells geben. Diese sind in den Tabellen 9.2 bis 9.5 dargestellt.

Genau wie in den vorliegenden Ergebnissen von Arena wird die optimale Konfiguration bei einer Maschinenverteilung von 1 Maschinen in der ersten Workstation, 1 Maschinen in der zweiten Workstation, 1 Maschinen in der dritten Workstation sowie 2 Maschinen in der vierten Workstation erzielt. Die einzelnen Größen für die drei Puffer betragen 2, 2 und 1.

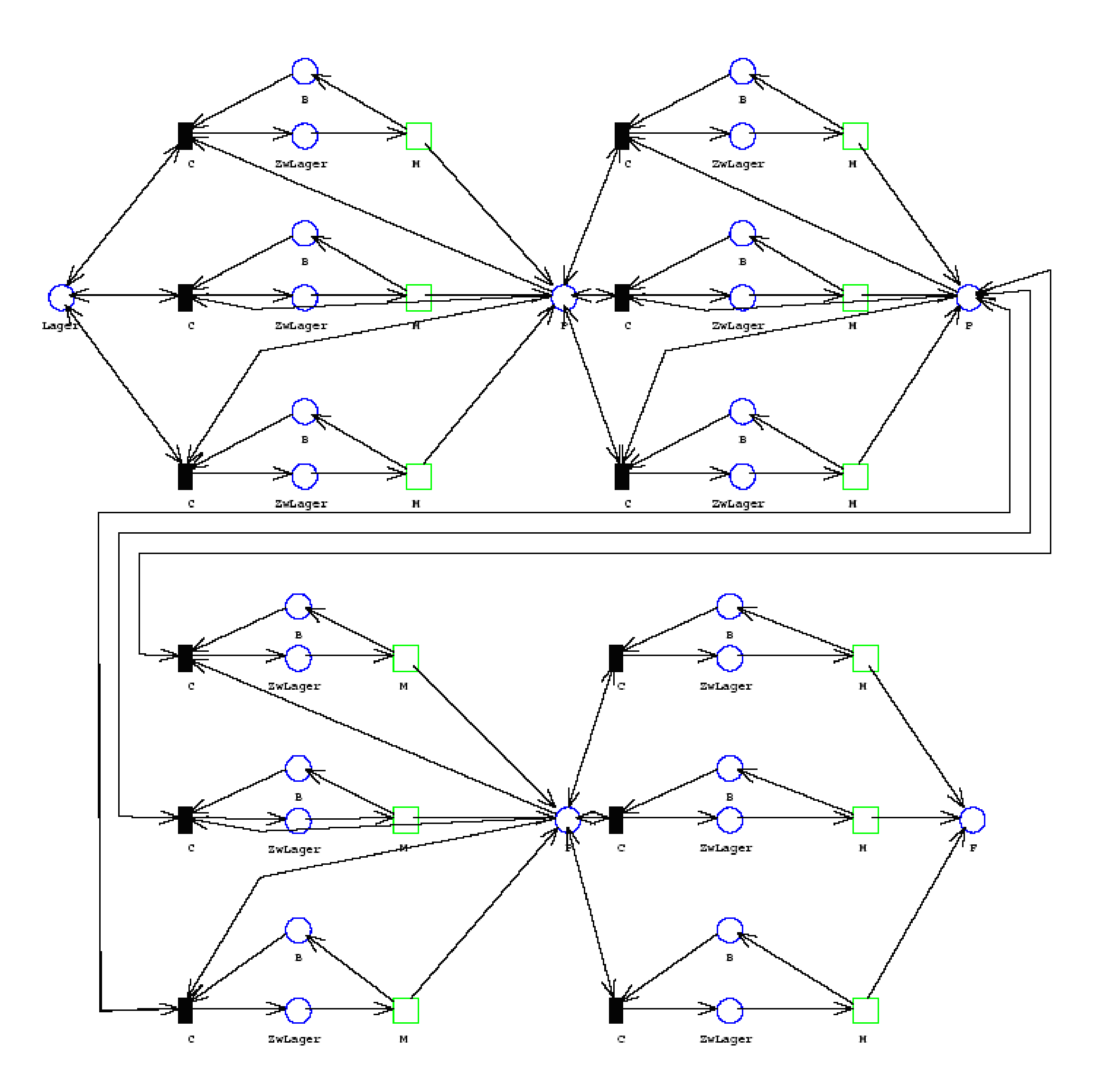


Abb. 9.17.: Vollständiges Modell

Konfiguration	1	2	3	4	5
Maschinen:					
Station 1	2	3	1	3	1
Station 2	3	2	1	3	1
Station 3	3	2	1	2	1
Station 4	2	3	1	2	2
Puffer:					
Puffer 1	10	5	1	8	2
Puffer 2	5	5	1	7	2
Puffer 3	10	5	1	4	1
Fertiggestellt:	4.059	2.974	1.092	4.309	4.401
Gewinn	$536.800	$329.800	$115.400	$592.800	$750.200

Tabelle 9.2.: Konfigurationsmöglichkeiten Teil 1

Konfiguration	6	7	8	9	10
Maschinen:					
Station 1	1	2	2	3	3
Station 2	3	2	1	3	3
Station 3	1	3	3	2	2
Station 4	3	3	3	2	2
Puffer:					
Puffer 1	10	6	5	10	5
Puffer 2	8	8	10	10	5
Puffer 3	6	6	4	10	5
Fertiggestellt:	2.214	2.935	1.484	4.454	4.185
Gewinn	$218.800	$317.000	$52.800	$610.800	$572.000

Tabelle 9.3.: Konfigurationsmöglichkeiten Teil 2

Konfiguration	11	12	13	14	15
Maschinen:					
Station 1	3	3	2	2	3
Station 2	2	1	3	2	2
Station 3	2	3	3	3	2
Station 4	1	3	2	3	3
Puffer:					
Puffer 1	5	4	10	8	8
Puffer 2	8	10	10	6	8
Puffer 3	6	4	10	4	8
Fertiggestellt:	2.733	1.484	4.084	2.966	2.988
Gewinn	$327.600	$28.800	$536.800	$325.200	$323.600

Tabelle 9.4.: Konfigurationsmöglichkeiten Teil 3

Konfiguration	16	17	18	19	20
Maschinen:					
Station 1	3	3	3	3	3
Station 2	3	3	3	3	3
Station 3	2	2	2	2	2
Station 4	2	2	2	2	2
Puffer:					
Puffer 1	8	7	9	10	7
Puffer 2	8	7	9	10	9
Puffer 3	8	7	9	10	5
Fertiggestellt:	4.420	4.350	4.442	4.420	4.380
Gewinn	$610.000	$599.000	$611.400	$604.000	$605.000

Tabelle 9.5.: Konfigurationsmöglichkeiten Teil 4

9.5. APNN-Modell einer Tankstelle

Christian Horoba, Peter Kissmann

In den folgenden Unterabschnitten soll auf die Anforderungen an die Modellierung einer Tankstelle mittels der APNN-Toolbox eingegangen, sowie deren Durchführung beschrieben werden.

9.5.1. Anforderungen

Die Aufgabe war, eine Tankstelle als Fluides Stochastisches Petri-Netz (kurz: FSPN) mit der APNN-Toolbox zu modellieren. Hierbei sollten folgende Anforderungen umgesetzt werden:

Die Tankstelle soll über vier Tanksäulen verfügen, an denen die Kunden den Treibstoff tanken können. Der Treibstoff wird von einem gemeinsamen Tankspeicher noch zu bestimmender Kapazität abgezogen. Sobald der Tankspeicher leer ist, wird die Tankstelle geschlossen und erst wieder geöffnet, wenn sich mindestens 100 Liter im Tankspeicher befinden. Damit sich der Tankspeicher wieder füllen kann, kommt ein Tankwagen gleichverteilt alle 6,75 bis 8,25 Stunden vorbei und füllt mit einer Rate von 300 Litern pro Minute insgesamt 900 Liter Treibstoff nach. Die Zwischenankunftszeiten der Autos sind exponentialverteilt und im Mittel wird alle 0,8 Minuten eines eintreffen. Bereits wartende Autos werden in dem Fall einer geschlossenen Tankstelle solange warten, bis sie wieder geöffnet wird; neuankommende Fahrzeuge hingegen werden vorbeifahren. Die Tankrate der Autos beträgt 20 Liter pro Minute.

Das Ziel war es nun, den Gewinn der Tankstelle zu maximieren. Als Eingabeparameter sollten die Tankspeichergröße und die Zwischenankunftszeiten des Tankwagens dienen. Die genaue Zielfunktion sowie die zugehörigen Parameter (etwa der Preis pro Liter gekauften oder verkauften Treibstoffs) waren noch festzulegen. Ebenso mußte noch die Wahrscheinlichkeitsverteilung der zu tankenden Treibstoffmenge der einzelnen Autos festgelegt werden.

9.5.2. Umsetzung

Leider war es nicht möglich, die Tankstelle als FSPN zu modellieren, da der Simulator erhebliche Fehler besaß. Daher wurde das gesamte Modell diskretisiert und als normales Stochastisches Petri-Netz mit der APNN-Toolbox modelliert. Das Ergebnis ist in Abbildung 9.18 dargestellt.

Leider kam es auch mit dem diskreten Simulator zu einigen Problemen, etwa mit den Inhibitor-Kanten; daher die Konstrukte mit solchen Stellen wie "AbgezogenerTreibstoff", an der die Treibstoffmenge gespeichert wird, die noch maximal in den Tankspeicher gefüllt werden darf.

Nun war es nötig, zunächst die Wahrscheinlichkeitsverteilung der zu tankenden Treibstoffmenge der einzelnen Autos festzulegen:

$$Prob\,(10\ Liter) = Prob\,(20\ Liter) = Prob\,(50\ Liter) = Prob\,(60\ Liter) = \frac{1}{8}$$

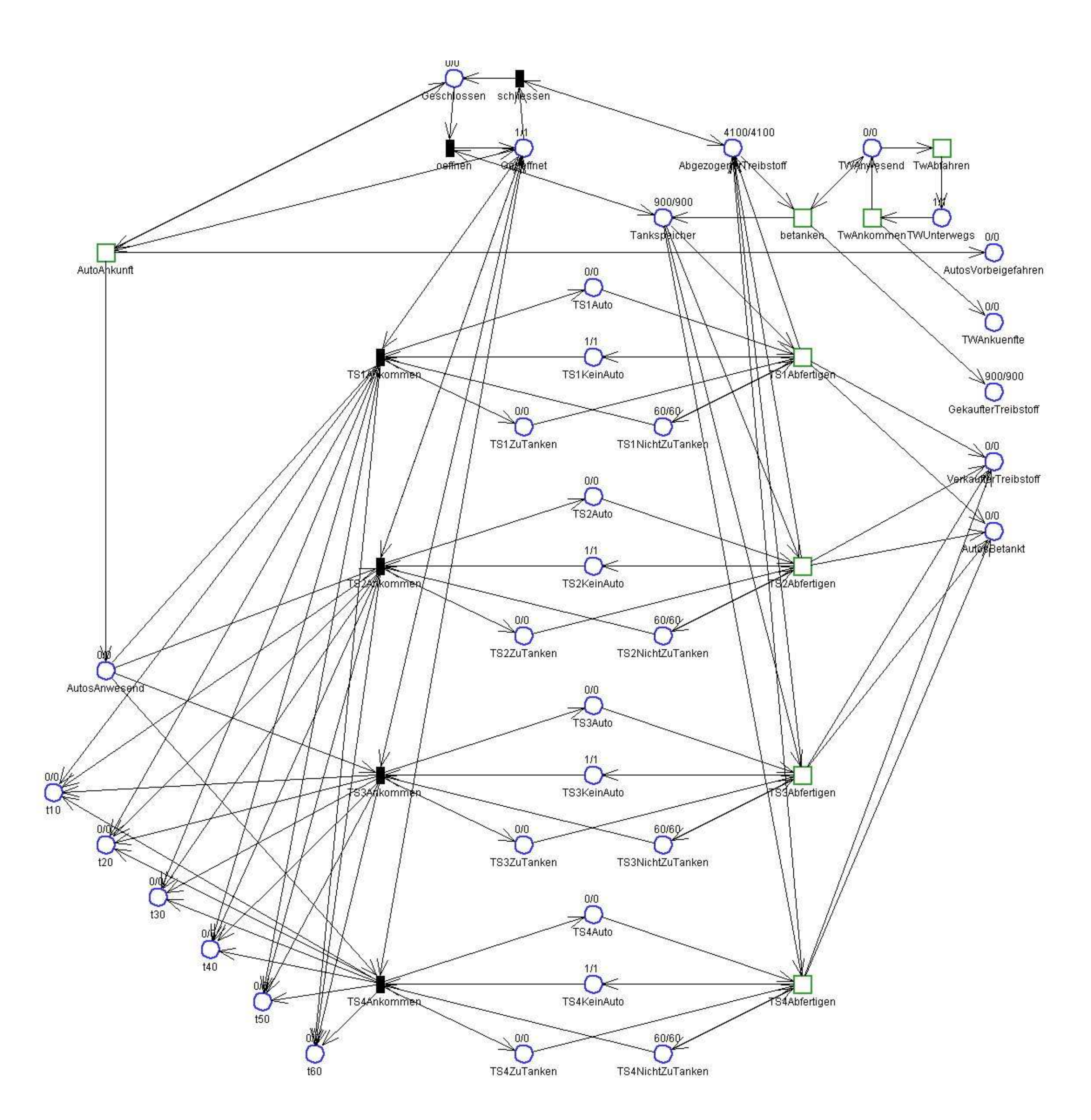


Abb. 9.18.: APNN-Modell einer Tankstelle

und

$$Prob\,(30\ Liter) = Prob\,(40\ Liter) = \frac{1}{4}.$$

Als letzter Schritt stand das Bestimmen einer sinnvollen Zielfunktion sowie der zugehörigen Parameter an:

Der Preis des zu verkaufenden Treibstoffs wurde auf 1 Euro gesetzt; der des einzukaufenden auf 0,50 Euro. Die Kosten pro Tankwagenankunft (unabhängig von der Menge gekauften Treibstoffes) wurden auf 112,50 Euro festgesetzt. Der Tankspeicher wurde als würfelförmig angenommen. Hier wurden die Kosten pro Quadratmeter Wandfläche auf 1000 Euro (= 10 Euro pro Quadratdezimeter) und die laufenden Kosten für Wartung, Versicherung und Miete auf 250 Euro pro Kubikmeter und Tag (= 0,25 Euro pro Liter und Tag) gesetzt.

Die Zielfunktion wurde folgendermaßen festgelegt: V sei das Volumen des Tankspeichers in Litern, Z die mittlere Zwischenankunftszeit des Tankwagens, t die Simulationszeit in Tagen, $g\,(V, Z, t)$ die gekaufte Menge Treibstoff in Litern, $v\,(V, Z, t)$ die verkaufte Menge Treibstoff in Litern, $i\,(V)$ die initiale Menge Treibstoff im Tankspeicher und $a\,(V, Z, t)$ die Anzahl der Tankwagenankünfte:

$$\begin{aligned} f\,(V, Z, t) &:= 1 \cdot v\,(V, Z, t) - 0,5 \cdot (g\,(V, Z, t) + i\,(V)) - 0,25 \cdot V \cdot t - \\ &\quad - (6 \cdot 10) \cdot \left(\sqrt[3]{V}\right)^2 - 112,5 \cdot a\,(V, Z, t) \\ &= v\,(V, Z, t) - 0,5 \cdot (g\,(V, Z, t) + i\,(V)) - 0,25 \cdot V \cdot t - 60 \cdot \left(\sqrt[3]{V}\right)^2 - \\ &\quad -112,5 \cdot a\,(V, Z, t) \end{aligned}$$

Für die Simulation wurde eine Modellzeit von 30 Tagen eingestellt; als Basiseinheit innerhalb des APNN-Modells wurden Sekunden gewählt, die Simulationszeit entspricht also 2592000 Einheiten. Zu Beginn jedes Simulationslaufs war der Tankspeicher stets mit 900 Litern Treibstoff gefüllt. Die Intervallbreite der Zwischenankunftszeiten des Tankwagens wurde so gewählt, daß der Variationskoeffizient der ursprünglichen Zufallsvariable beibehalten wurde.

9.5.3. Response Surface

Mit obiger Zielfunktion ergab sich die in Abbildung 9.19 abgebildete Ausschnitt der Response Surface.

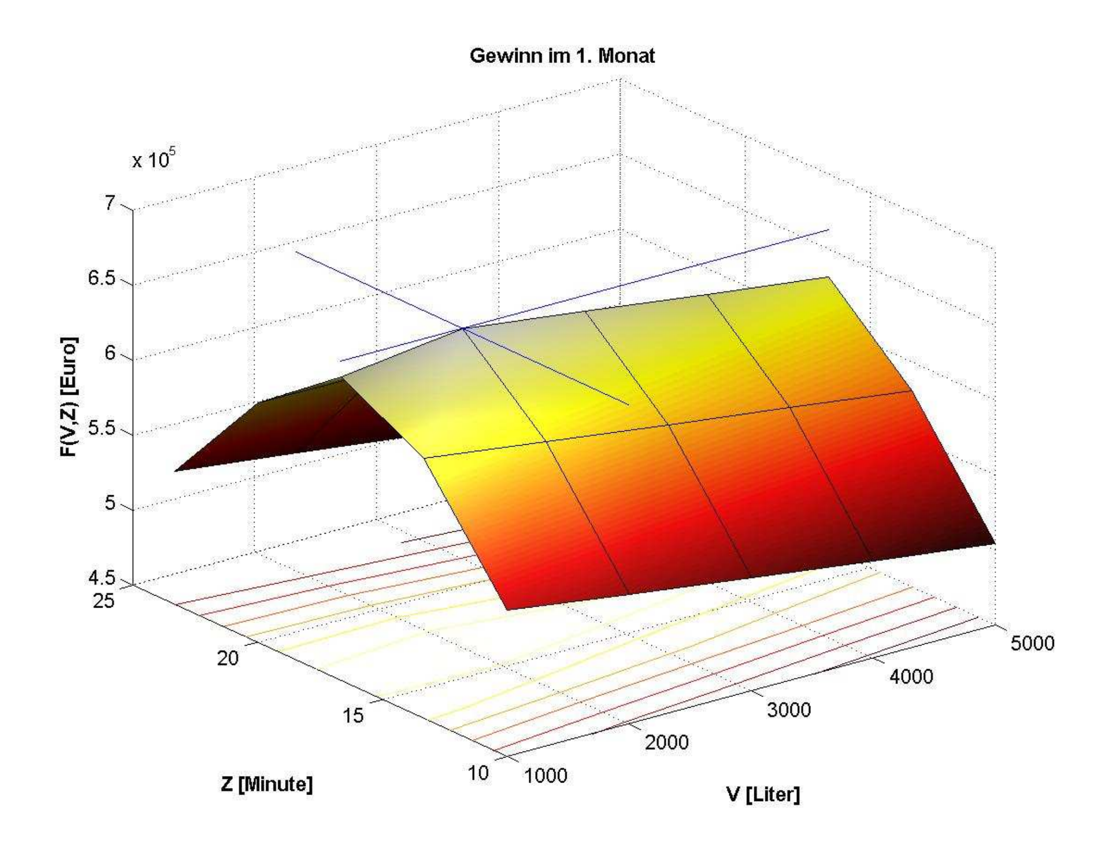


Abb. 9.19.: Response-Surface des APNN-Modells einer Tankstelle

9.6. Fertigungsstraße

Thomas Hutter

Die Realisierung dieses Modells erfolgt nicht mit Hilfe des APNN- oder ProC/B Toolsets, sondern wurde in C++ programmiert. Im Kapitel 11.4.2.4 wird dieses Plugin noch genauer erläutert, so daß an dieser Stelle nur eine kurze Einleitung des Modells erfolgt.

Das Model der Fertigungsstraße besteht aus n- hintereinander geschachtelte Bedienstationen, wobei jede Station eine Kapazität von 10 Einheiten aufweist. Die Einstellung für die Variable n wurden auf einen Startwert von 2 gesetzt, ist aber jederzeit änderbar. Mit einer Ankunftsrate von 0.5 gelangen Kunden in die erste Bedienstation (Abbildung 9.20).

Nach erfolgreicher Bedienung in der ersten Station wird der Kunde an Station 2 weitergereicht und anschließend verläßt er das Model. Ziel hierbei war es den Gewinn zu maximieren, indem die Bedienrate möglichst geschickt gewählt wird. Für eine genauere Beschreibung der einzelnen Bedienraten, sowie deren Einfluß auf die Optimierung sei auf Kapitel 10.1 verwiesen.

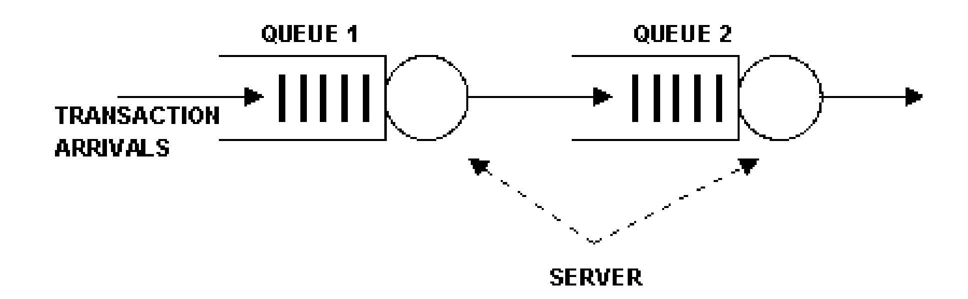


Abb. 9.20.: Fertigungsstraße

9.7. (s,S)-Lagersystem

Thomas Hutter

Dieses Model wurde in C++ programmiert. Im Kapitel 11.4.2.3 wird dieses Plugin noch genauer erläutert, so daß hier nur eine kurze Erläuterung des Modells stattfindet.

Das (s,S)-Lagersystem beschäftigt sich mit einer optimalen Lagerhaltungsstrategie. Dabei wird einmal pro Monat, genau gesagt am ersten jeden Monats, eine Bestellung rausgegeben. Die bis dato sich im Lager befindlichen Waren müßen also die gesamten Kundenbestellungen decken können. Um eine mögliche Entscheidungsfrage bezüglich einer Bestellung zu klären, dient Abbildung 9.6

$x < s$	Waren bestellen
$x >= s$	keine Ware bestellen

Tabelle 9.6.: Entscheidungsfrage

Das s definiert eine vom Benutzer festgelegte Größe, die es zu Optimieren gilt. Doch genaueres ist dem Kapitel 10.2 zu entnehmen. Tritt der erste Fall in Kraft, stellt sich die Frage, wie viel Ware muß bestellt werden. Die Menge setzt sich aus der Differenz von S und x zusammen, wobei S eine Grenze angibt, bis zu welcher Menge Waren im Lager gehalten werden sollen. Auch diese Größe ist variabel und gilt es zu Optimieren. Die Lieferung der bestellten Ware erfolgt zufällig verteilt zwischen 2 und 4 Wochen. Ebenso die Anfragen der Kunden unterliegt einer Zufallsverteilung mit vorgegebener Rate, wie sie in Tabelle 9.7 zu sehen ist.

Inwiefern die einzelnen Kosten Einfluß auf das (s,S)-Lagersystem haben, wird im Kapitel 10.2 erläutert.

9.8. Mix Kaskade

Thomas Hutter

Beim surfen hinterlässt jeder Nutzer Datenspuren, die eine personenbezogene Rekonstruktion des Surfverhaltens ermöglichen. Diese Informationen fallen beispielsweise beim

Rate	Anzahl
$\frac{1}{6}$	1 Stück
$\frac{1}{3}$	2 Stück
$\frac{1}{3}$	3 Stück
$\frac{1}{6}$	4 Stück

Tabelle 9.7.: Anfragen der Kunden

Provider oder mithörenden Dritten an. Anonymität im Internet heißt, für alle anderen, ausgenommen der Kommunikationspartner nicht identifizierbar zu sein. Zum einen können die Inhalte der ausgetauschten Nachrichten mit Hilfe von Verschlüsselung geheim gehalten werden. Zum anderen sollte man durch die Nutzung einer wirksamen Anonymisierung schon die Entstehung von Verbindungsdaten vermeiden, die Aufschluß darüber geben, mit wem man kommuniziert hat. Somit bleibt verborgen, welche Internetseiten besucht und welche Anfragen an Datenserver gestellt werden. Die einfachsten Anonymisierer sind so genannte Proxy-Server, die eine Internetanfrage stellvertretend für den Nutzer an den Server weiterleiten. Bei dieser Grundform von Anonymisierung, ist die Kommunikation aber sowohl gegenüber dem Provider als auch gegenüber dem Betreiber des Proxy selbst nicht mehr anonym. Besser geeignet sind echte Anonymisierungs-Architekturen, wie z.B die Mix-Kaskade, das durch die verschlüsselte Weiterleitung von Nachrichten zwischen den Mix-Stationen eine Reidentifizierung der Kommunikation nur noch sehr schwer möglich macht.

Die Realisierung der Mix-Kaskade erfolgte mit Hilfe der APNN-Toolbox. Das Modell besteht im groben aus einem Client, drei Mix-Klassen und einem Server (Abbildung 9.21).

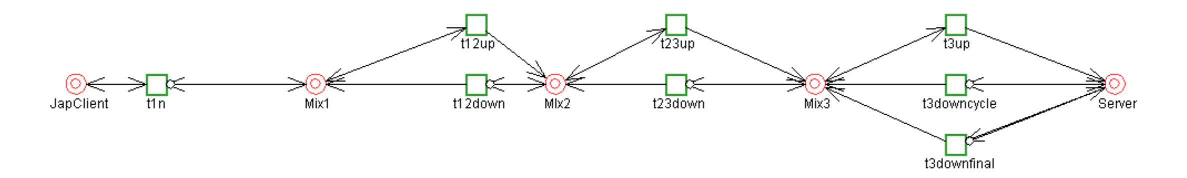


Abb. 9.21.: Mix Kaskade Übersicht

Der Client sorgt für eine zufällige Ankunftsrate von Datenanfragen an den Server. Hierfür wurden verschiedene Verteilungen hintereinander geschachtelt und an die erste Mix-Klasse weitergereicht (Abbildung 9.22).

Erreicht eine Anfrage die erste Klasse, wird überprüft, ob genügend Pufferkapazität vorhanden ist und in der Transition *mixqueue* zwischengelagert. Andernfalls wird die Anfrage verworfen. Sobald genügend CPU-Leistung für die Upstreamcodierung vorhanden ist, wird die Anfrage verschlüsselt und an die zweite Klasse weitergereicht. Die Entschei-

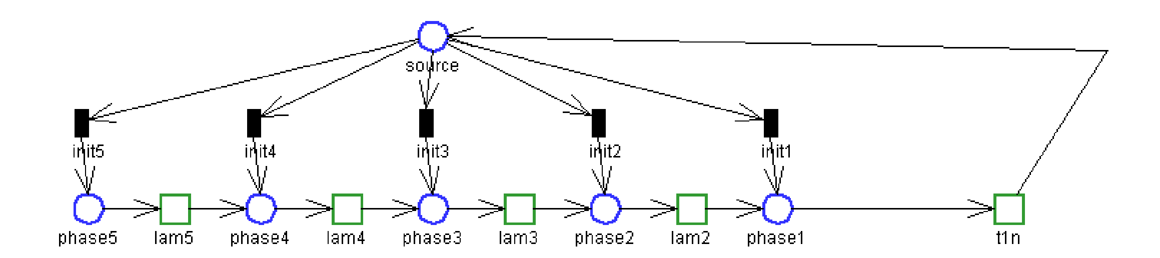


Abb. 9.22.: Mix Kaskade Client

dung für die CPU-Leistung wird in der Transition *M1getCPUup* getroffen. Mit einer Wahrscheinlichkeit von p wird die Leistung an *M1getCPUup* und mit einer Wahrscheinlichkeit von $1 - p$ an *M1getCPUdown* gegeben. Folglich haben diese beiden Werte einen sehr starken Einfluß auf den Durchsatz des Modells. Welche Werte jedoch sinnvoll sind, wird im Kapitel 10.3 genauer untersucht. Da der Aufbau aller drei Klassen identisch ist, wird auf eine weiterführende Erklärung der anderen Klassen verzichtet (Abbildung 9.23).

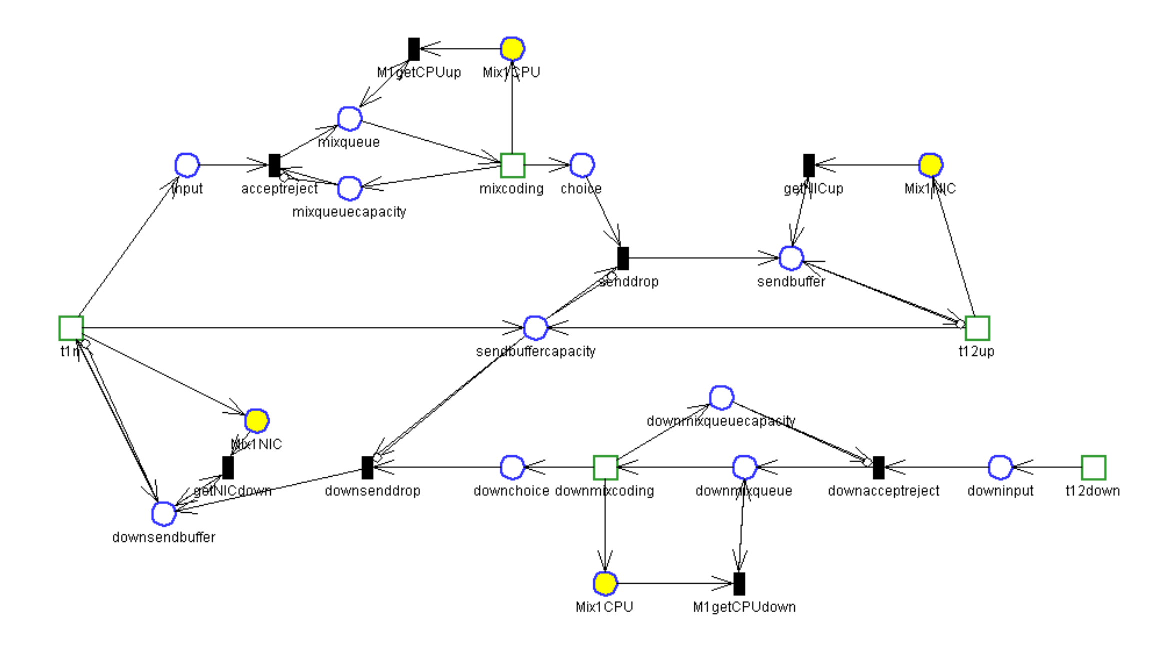


Abb. 9.23.: Mix-Klasse 1

Nach Durchlauf aller Mix-Klassen gelangt die Anfrage zum Server. Dort wird sie bearbeitet und der angeforderte Downstream wird losgeschickt. Die Pakete einer Anfrage werden mit Hilfe von *t3downcycle* an die Mix-Klasse 3 geleitet, wobei das letzte Paket durch die Transition *t3downfinal* läuft. Auch hier muß zunächst der Downstream alle drei

Klassen durchlaufen, wobei die Verschlüsselung wieder rückgängig gemacht wird, damit am Ende der ersten Klasse eine eindeutige Zuordnung geschehen kann (Abbildung 9.24)

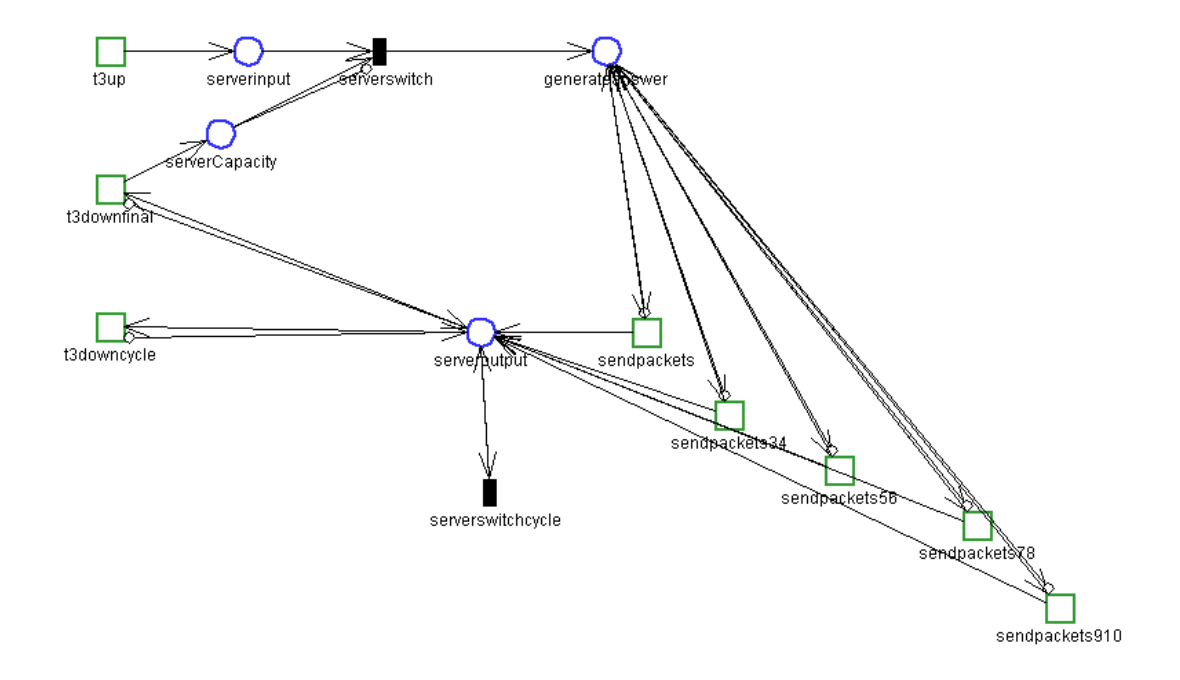


Abb. 9.24.: Mix Kaskade Server

9.9. Supply Chain

Thomas Hutter

Supply Chain ist die Planung und Steuerung der Logistikkette eines produzierenden Unternehmens von seinen Lieferanten bis zu seinen Kunden. Häufig sogar bis zum Endverbraucher. Aus Sicht eines produzierenden Unternehmens existieren entlang dieser Kette diverse Stationen, die dabei eine Rolle spielen könnten: Lieferanten, Beschaffungslager, Logistikdienstleister und Dienstleister, Speditionen, Distributionslager und Verteilzentren, Groß- und Einzelhandel, Endverbraucher und weitere. Unternehmensübergreifende Zielkonflikte treten auf, wenn beispielsweise der Abnehmer seine Bestellmengen absenkt und damit verbunden die Anzahl der Auftragsabwicklungs- und Transportprozesse beim Lieferanten steigt. Auf der anderen Seite kann bei einer Bestandsminimierung beim Lieferanten der Abnehmer gegenüber seinen Kunden in Lieferschwierigkeiten geraten, wenn durch diese Bestandsminimierung der Warennachschub nicht mehr sichergestellt ist. Folglich geht es im Supply Chain einzig und allein darum, die einzelnen Glieder dieser Kette so zu planen und zu steuern, daß die Produkt- und Warenschlangen möglichst klein werden, der jeweilig Kunde möglichst zufrieden, sprich optimal versorgt wird und man selber als produzierendes Unternehmen dieses möglichst kostenminimal bewerkstel-

ligt. Der Ablauf der Supply Chain wurde mit Hilfe des Toolset Proc/B modelliert. Die Abbildung 9.25 zeigt die Hierarchie des Modells in solch einer Notation.

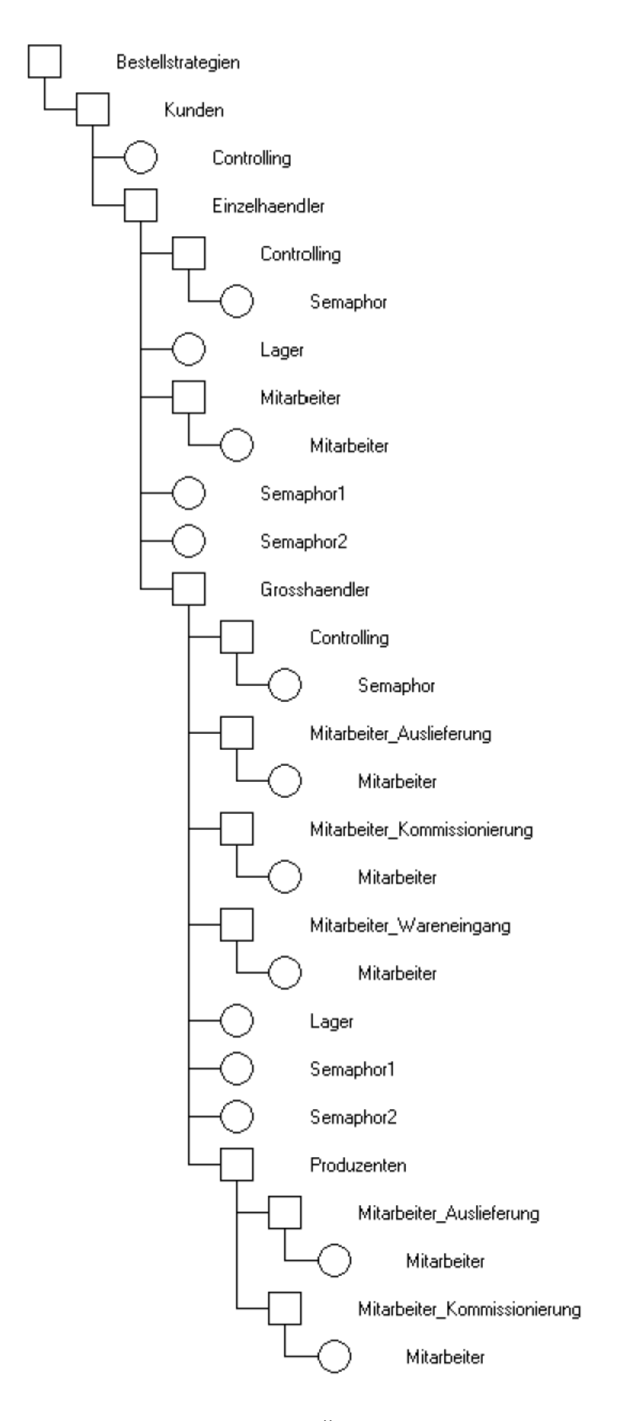


Abb. 9.25.: Übersicht

Es ist so angelegt, daß die jeweiligen Lieferanten mittelbare oder unmittelbare Unter-

einheiten der Empfänger einer Lieferung sind. Somit stehen nach der Umwelt (*FE Bestellstrategien*) die Kunden auf oberster Hierarchieebene gefolgt von der Einzelhändler-, Großhändler- und Produzentenebene (FE=Funktionseinheit). Diese Konstruktion vereinfacht den Aufruf der Dienste und spiegelt die reale Dienstleistungsfunktion wider.

Die *FE Bestellstrategien* (Abbildung 9.26) steht auf der obersten Ebene der *FE-Hierarchie* und bildet die Schnittstelle zur Umwelt. 10 Kundenklassen sind definiert, für die jeweils eine eigene Prozesskette beschrieben ist, damit auf einfache Weise Variationen der Parameter einzelner Kundenklassen vorgenommen werden können. Jeder Prozess einer solchen Kette repräsentiert einen Kunden der entsprechenden Kundenklasse.

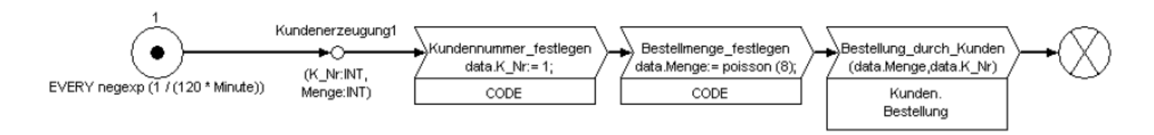


Abb. 9.26.: FE-Kunde

Die *FE Kunden* (Abbildung 9.27) verfügt über eine Prozesskette, die den Bestellvorgang der Kunden beschreibt. Dazu wird die Prozessausführung einer Bestellung entsprechend an den zugeordneten Handelspartner durch einen Prozessaufruf weitergeleitet. Ist der Handelspartner des Kunden nicht in der benachbarten Ebene (dem Einzelhändler), so wird dabei ein Dienst der benachbarten Ebene aufgerufen, der lediglich der Weiterleitung dient. Die Zuordnung geschieht mit Hilfe von globalen Variablen. Dazu hat der Modellierer vor dem Start der Analyse für jede Kundenklasse den Typ des Handelspartners ("e" für einen Einzelhändler, "g" für einen Großhändler und "p" für einen Produzenten) sowie eine Nummer des Partners in je einem Array festzulegen. Diese Arrays sind sortiert nach der Nummer der Kundenklasse, die jeder Prozess der *FE Kunden* als Parameter mitführt. So kann jeder Prozess mit Hilfe seiner eigenen Nummer auf die Stellen der Arrays zugreifen, die die Zuordnung zu seinem Handelspartner eindeutig beschreiben. Da es sich bei den in der Array- Definition enthaltenen Werte lediglich um Default-Werte handelt, ist für zukünftige Betrachtungen eine dynamische Modifikation der Werte und dementsprechend der Handelspartner-Zuweisung problemlos möglich.

Nach erfolgter Bearbeitung der Bestellung wird die Prozesskontrolle wieder an die *FE Kunden* zurückgegeben, die durch Benutzung der *FE Controlling* die Bestellung als erfolgreich oder nicht-erfolgreich (d.h. der Handelspartner war nicht lieferfähig) dokumentiert. Allerdings wird hier nur auf die *FE Einzelhaendler* eingegangen, da sich die Optimierung im Kapitel 10.4 auch nur mit dieser beschäftigt.

Die *FE Controlling* (Abbildung 9.28) dient zur Erfassung des Ergebnisses der Bestellvorgänge, um somit nach einer Simulation Aussagen über den Grad der Erfüllung der Kundenwünsche treffen zu können. Dies soll anhand einer Bewertung hinsichtlich des Erfolges einer Bestellung (Kundenwunsch erfüllt/nicht erfüllt) sowie der umgesetzten Verkaufsmengen und der entgangenen Verkaufsmengen bei Nicht- Lieferbarkeit erfolgen.

Die *FE Einzelhaendler* bietet drei Dienste nach außen an (Abbildung 9.29). Zwei

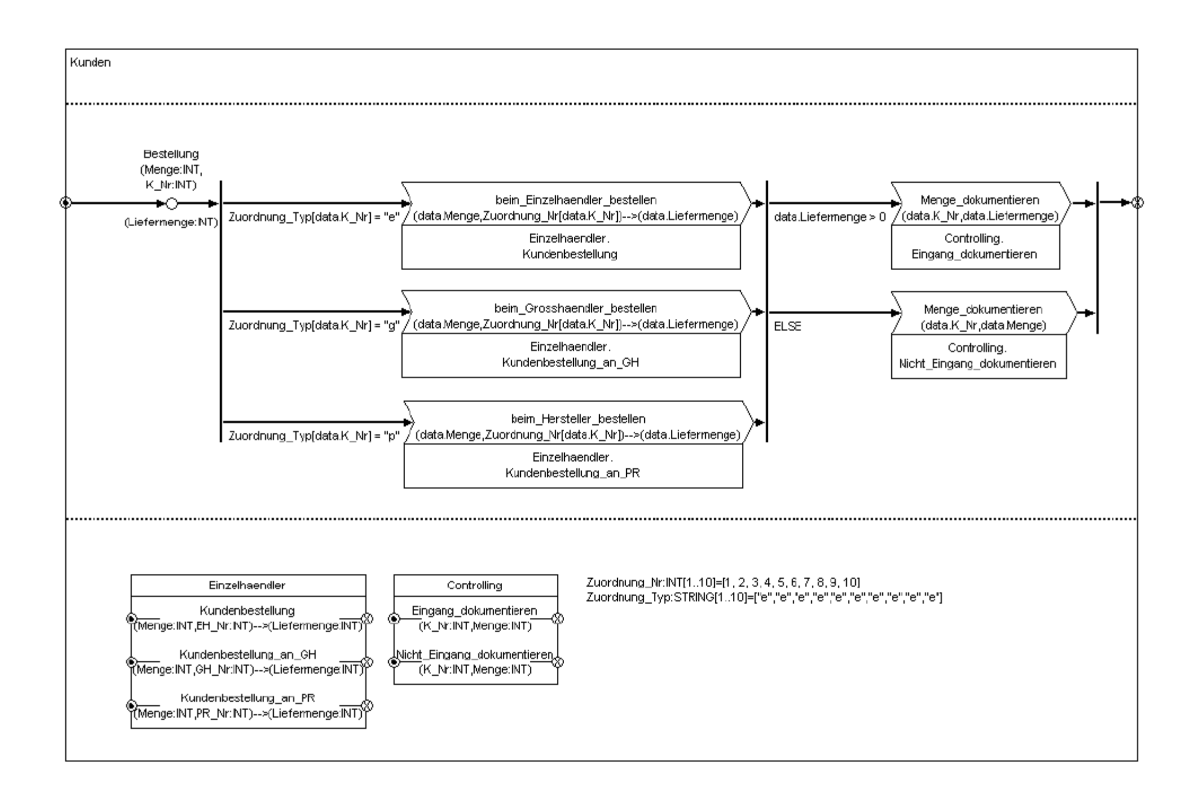


Abb. 9.27.: FE-Bestellstrategie

davon dienen lediglich dem Durchschleifen von Bestellungen eines Kunden an einen Großhändler oder Produzenten, bei denen kein Einzelhändler Handelspartner ist. Der Dienst Kundenbestellung verarbeitet die an Einzelhändler gerichteten Bestellungen und initiiert, falls nötig, unter Berücksichtigung der gewählten Strategien, Parametern und Lagerbeständen eigene Bestellungen an den zugehörigen Handelspartner. Sollte dies der Fall sein, werden die angeforderten Mengen bestellt und bei Lieferung eingelagert, damit der Kunde beliefert werden kann.

Somit ist ein Kreislauf von der Kundenbestellung bis hin zur Warenbereitstellung beschrieben. Dabei sei nochmal angemerkt, das hier nur die Bestellstrategie über die Einzellhändler erläutert wurde.

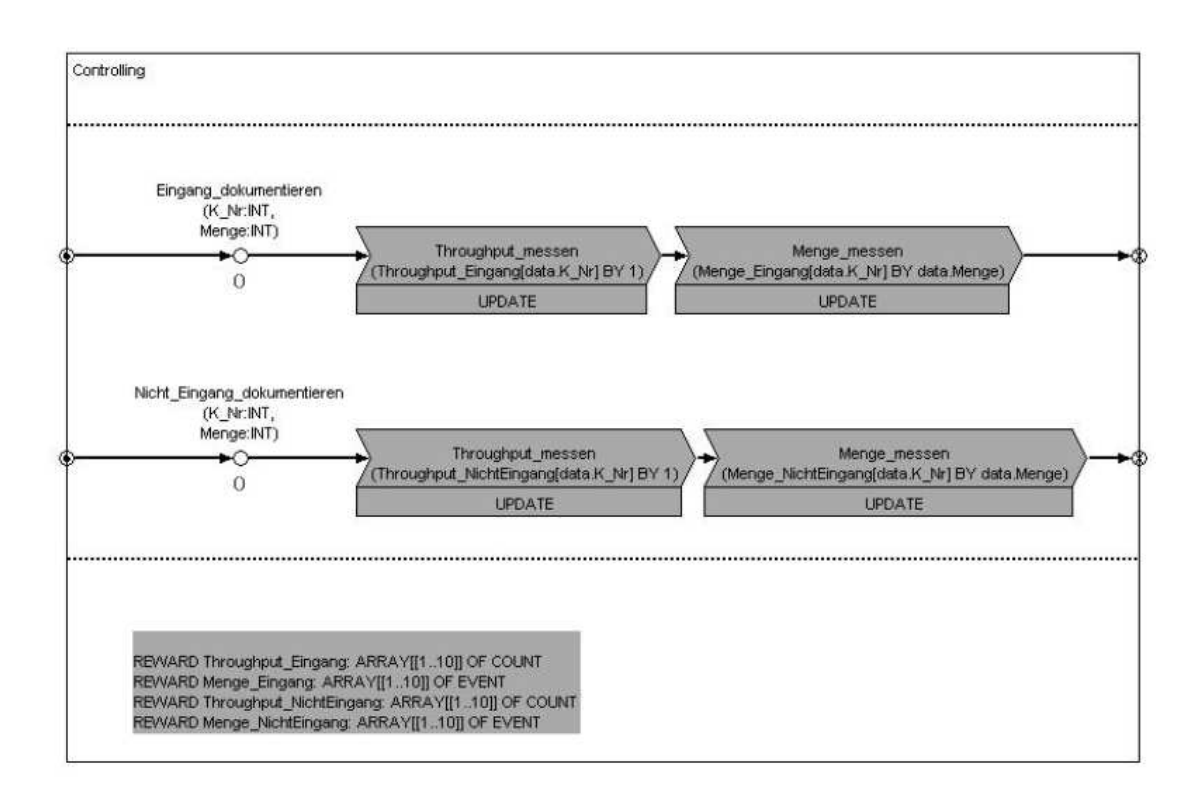


Abb. 9.28.: FE-Controlling

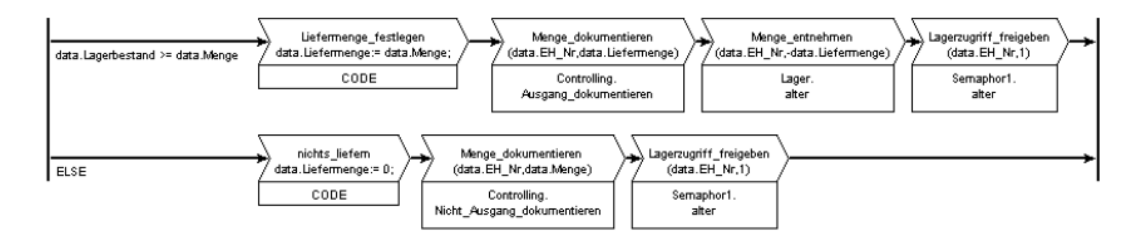


Abb. 9.29.: FE-Einzelhaendler

10. Optimierung von Simulationsmodellen

Thomas Hutter

10.1. Fertigungsstraße

Es wurde versucht in diesem Model die Bedienrate beider Stationen zu optimieren. Die Gewinnfunktion ist in Formel 10.1 zu sehen.

$$\text{Gewinn} = \frac{10000 \cdot \text{Anzahl Kunden}}{\text{Kosten} \cdot \text{Modellzeit}} - 400 \tag{10.1}$$

Dabei sei erwähnt, daß es sich um ein Maximierungsproblem handelt und die Bedienrate direkten Einfluß auf den Zähler der Funktion hat. Somit sollte sie möglichst klein gewählt werden. Die auftretenden Kosten steigen exponentiell im Verhältnis zur steigenden Bedieneranzahl. Eine detailliertere Auflistung ist aus der Tabelle 10.1 zu entnehmen.

Station 1	1*Bedienrate=Kosten
Station 2	5*Bedienrate=Kosten
Station 3	9*Bedienrate=Kosten
Station 4	13*Bedienrate=Kosten

Tabelle 10.1.: Bedienkosten pro Station

Auch dieses Problem wurde mit Hilfe von *RSM* gelöst. Der Suchbereich lag zwischen 0 und 2 und der vorgegebene Startpunkt bei $[1;1]$. Der Verlauf des Algorithmus' während der Optimierung ist in Abbildung 10.1 sehr schön dargestellt.

Nach 206 Responseaufrufen terminierte *RSM* und lieferte einen Responsewert von $[0.60958; 0.446619]$ (Abbildung 10.2).

Um sich einen besseren Eindruck des gefundenen Optimums im Verhältnis zur gesamten Response Surface zu verschaffen ist die Grafik 10.3 sehr gut geeignet. Hier erkennt man, das es wirklich nur ein Maximum gibt, welches von *RSM* gefunden wurde und somit das *Fertigungsstraßen*-Problem gelöst wurde.

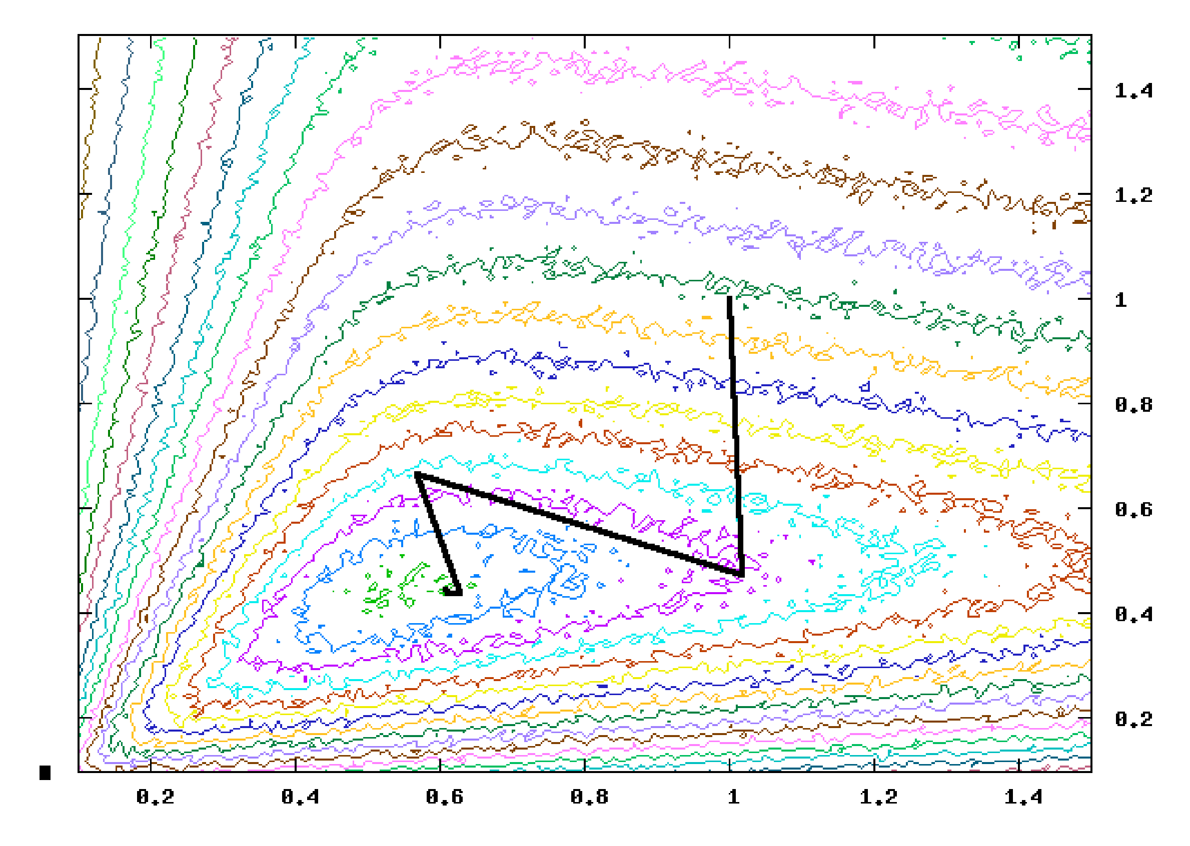


Abb. 10.1.: Plot der Abstiegsrichtung für die Fertigungsstraße

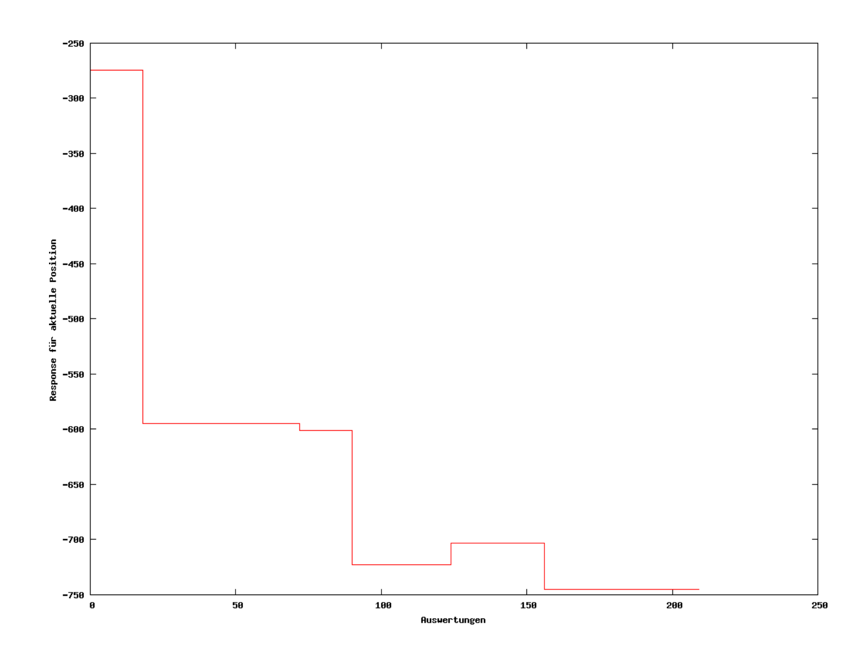


Abb. 10.2.: Anzahl der Auswertungen für die Fertigungsstraße

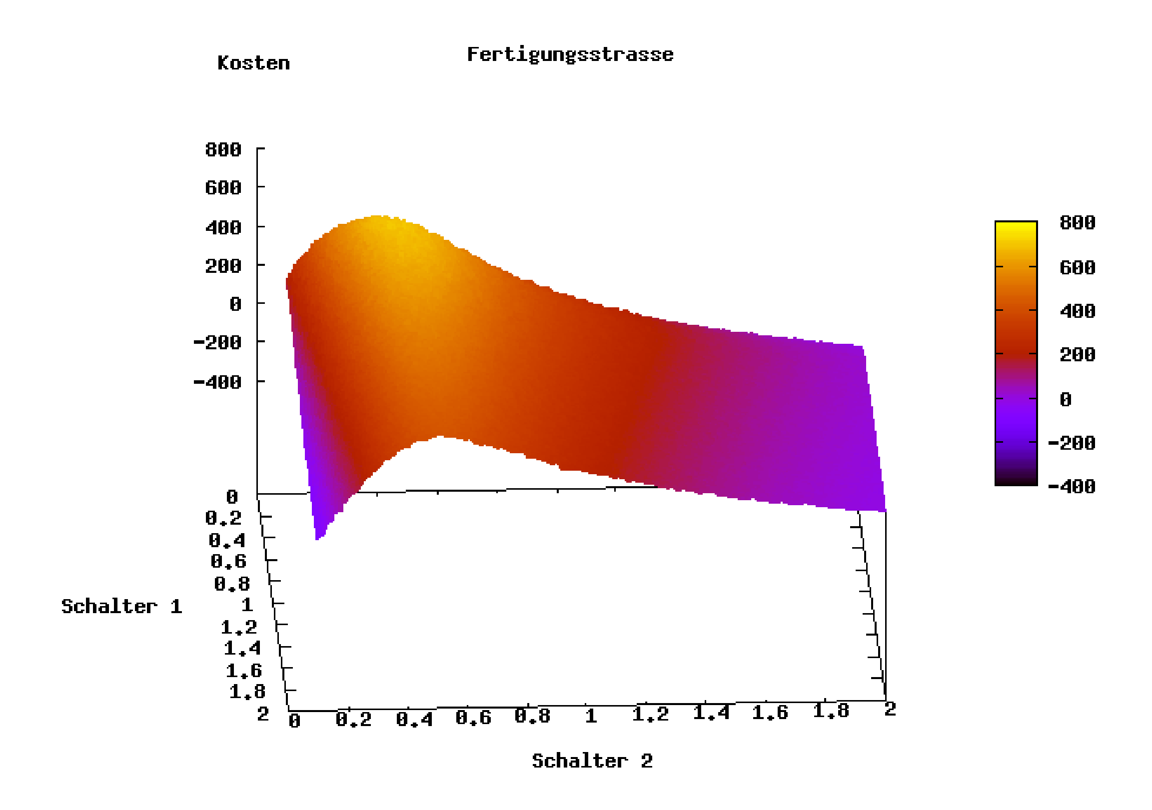


Abb. 10.3.: Responsefläche Fertigungsstraße

10.2. (s,S)-Lagersystem

Hier wurde versucht, eine möglichst kleine Konfiguration von s und S zu finden, damit die Kosten minimal werden. Die gesamten Kosten setzen sich aus sogenannten Strafkosten, Lagerkosten und Bestellkosten zusammen. Strafkosten treten immer dann auf, sobald eine Kundenanfrage eintrifft, welche eine bestimmte Menge verlangt, diese aber aufgrund eines unzureichenden Inventarbestandes nicht bedient werden kann. Da die Kunden aber in dem Model immer bedient werden, wird der Lagerbestand negiert und es fallen Kosten in Höhe von 5 Euro pro Monat und Stück an. Lagerkosten belaufen sich auf 1 Euro pro Monat und Stück und die Bestellkosten setzen sich aus 32 Euro Fixkosten und das Dreifache der Menge für die variablen Kosten zusammen. Das Minimierungsproblem wurde mit Hilfe des *RSM* Optimierers gelöst. Er wurde am Punkt $[1; 1]$ gestartet und der gesamte Suchbereich befand sich zwischen $[0; 120]$. Der erstellte Plot in Abbildung 10.4 zeigt den Verlauf des *RSM* Algorithmus' während der Optimierung. Dabei ist auf der X-Achse die Einstellung für s und auf der Y-Achse die von S aufgetragen. Hier ist ganz deutlich zu erkennen, daß die Optimale Konfiguration für (s, S) im Bereich von $(22, 58)$ liegt.

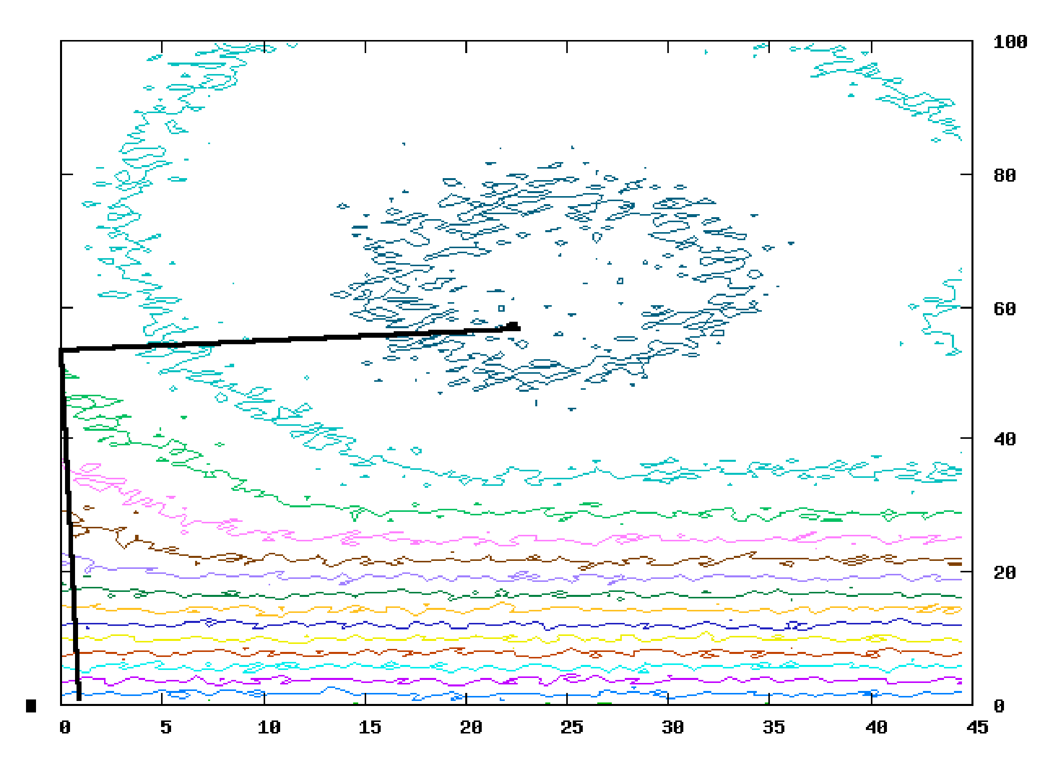


Abb. 10.4.: Plot der Abstiegsrichtung für das (s,S)-Lagersystem

Die Anzahl der Auswertungen betrug 186 Responseaufrufe, wobei nach ca. 92 keine Verbesserung mehr auftrat (Abbildung 10.5). Nach erfolgreicher terminierung des Algorithmus' wurde als Minimum der Wert $[22.4757; 57.4465]$ zurückgeliefert.

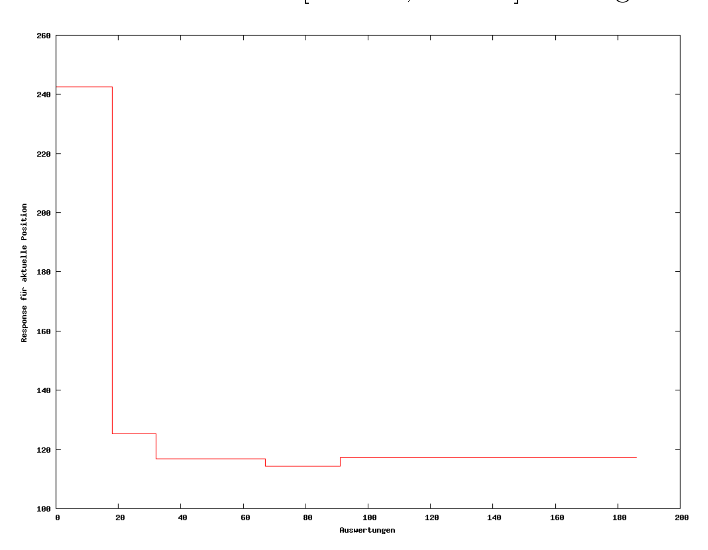


Abb. 10.5.: Anzahl der Auswertungen für das (s,S)-Lagersystem

Um sich eine genauere Vorstellung der Response Surface zu machen, dient die Grafik

10.6. Hier ist nochmal das Minimum gegenüber der gesamten Fläche zu erkennen. Somit lieferte *RSM* einen sehr schönen Wert.

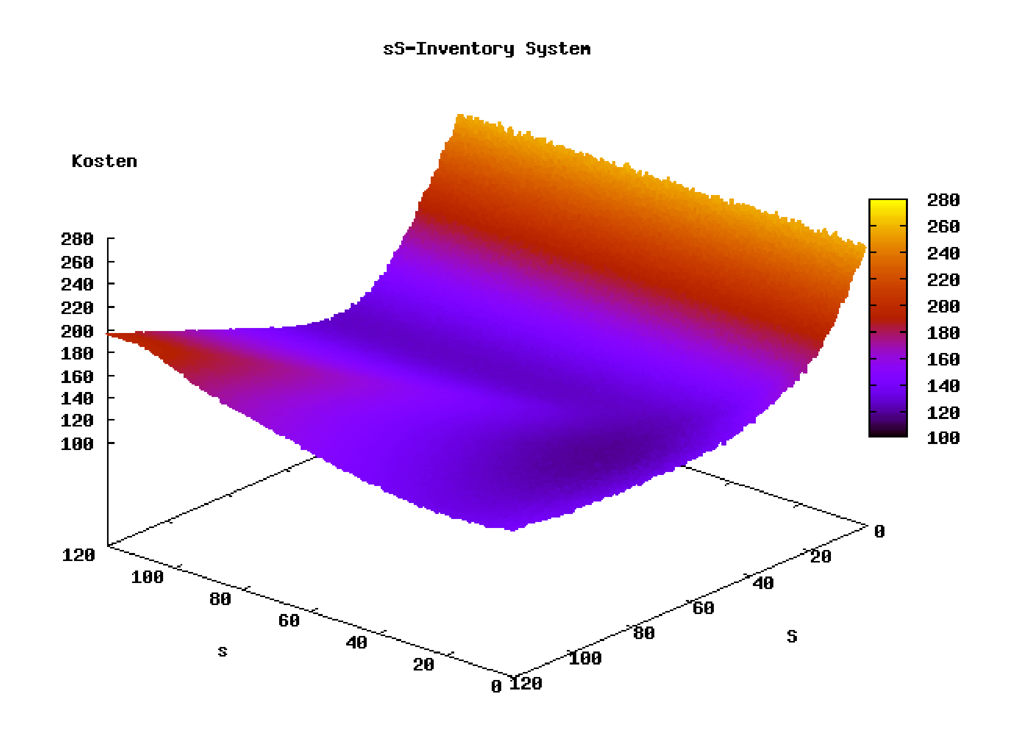


Abb. 10.6.: Responsefläche (s,S)-Lagersystem

10.3. Mix Kaskade

Bei diesem Model wurde versucht den Durchsatz der ankommenden Downstreams zu maximieren. Dabei wurde an den Transitionen *M*getCPUup* und *M*getCPUdown*in den einzelnen Mix-Klassen die Wahrscheinlichkeiten variiert. Diese haben Einfluß darauf, ob ankommende Anfragen akzeptiert oder verworfen werden. Das $*$ steht hierbei für die einzelnen Mix-Klassen. Bei der Optimierung wurde eine Konfiguration für alle Mix-Klassen gewählt. Desweiteren sollte darauf geachtet werden, daß alle Puffer mindestens zu 50% gefüllt sind. Andernfalls wurde vom ermittelten Durchsatz ein Strafwert subtrahiert. Da das Model ursprünglich ein Maximierungsproblem darstellt, wurde die Responsefunktion negiert, wodurch ein Minimum gesucht wird. Die Einstellung sollte zur Anonymität der Nutzer beitragen. Hier waren zwei Puffer relevant. Zum einen *mixqueuecapacity* und zum anderen *downmixqueuecapacity*. Beide befinden sich ebenfalls in den einzelnen Mix-Klassen und erhielten identische Werte für die Kapazität. Somit ergibt sich ein 2-dimensionales Optimierungsproblem. Mathematisch ausgedrückt, ergibt sich die Formel 10.2 für unseren Responsewert, wobei $p1$ die Wahrscheinlichkeit von *mixqueu-*

ecapacity und $p2$ die Pufferkapazität darstellt. Die Variable b ist der gemessene Durchsatz in der Transition *t1n*, wobei nur der Durchsatz des Downstreams gezählt wurde.

$$\max Z(p1, p2) = \text{Durchsatz}(\text{Modell}(p1, p2)) - \left(\frac{p2}{2} - \varnothing b\right)^2 \tag{10.2}$$

Die Optimierung erfolgte sowohl mit dem *RSM* als auch mit dem *EA* Algorithmus. Die Wahrscheinlichkeiten lagen im Intervall von $[0,1; 09]$ mit einer Schrittweite von $0,1$. Die Puffergröße variierte von 100 bis 1.000 und wurde in 100-er Schritten erhöht.

In der Grafik 10.7 ist die ermittelte Responseoberfläche für verschiede Konfigurationen zu sehen.

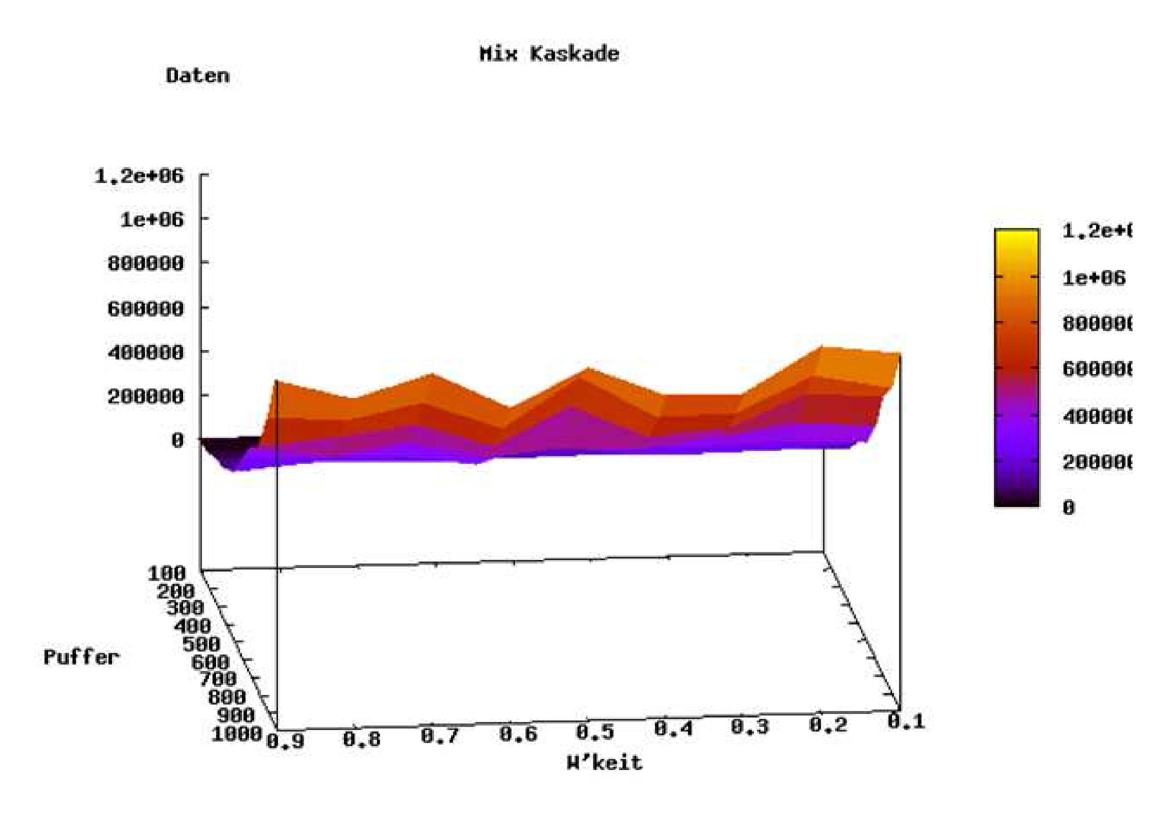


Abb. 10.7.: Responsefläche Mix Kaskade (großer Puffer)

Hier fällt auf, das bei größerer Pufferkapazität auch der Durchsatz steigt. Dieser Effekt ist dadurch zu erklären, daß bei steigender Puffergröße auch weniger Anfragen verworfen werden und somit eine höhere Anzahl zwischengelagert werden kann. Betrachtet man aber die einzelnen Wahrscheinlichkeiten, ist deutlich zu erkennen, daß der Responsewert sein Minimum im Bereich von $0,3$ bis $0,4$ erreicht (Abbildung 10.8).

Der *RSM* Algorithmus lieferte schon nach ca. 100 Responseaufrufen einen Wert von $0,3819$. Die Anzahl der *STOPPINGSTEPS* wurde auf 2 gesetzt und die Werte für *DECREASECOUNT* und *DECREASEFACTOR* waren jeweils 1 und 20.

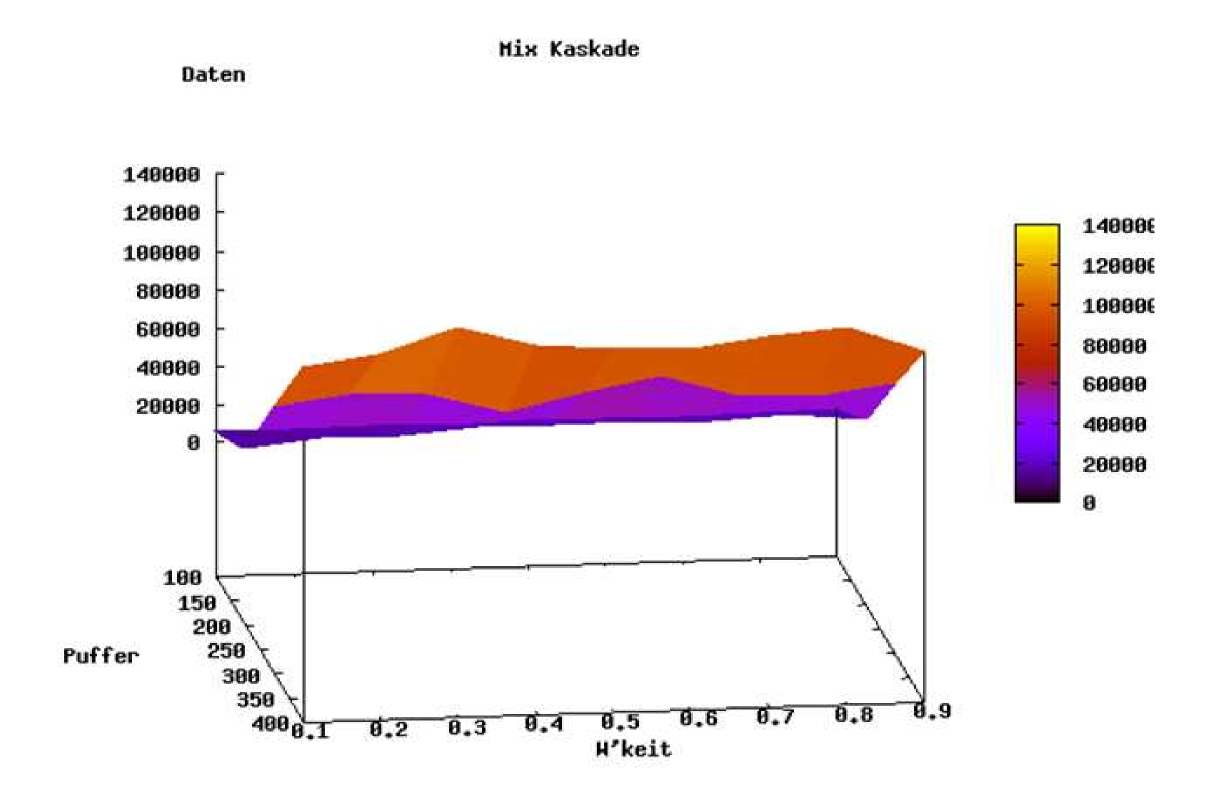


Abb. 10.8.: Responsefläche Mix Kaskade (kleiner Puffer)

Bei *EA* wurde ein Minimum von 0, 3672 erreicht. Auch dieser Wert liegt sehr gut im Bereich des Optimums für kleine Pufferkapazitäten. Beim durchlauf des Algorithmus' wurde eine *PPOPULATIONSIZE* von 10 und eine *NUMBEROFGENERATIONS* von 5 gewählt. Für eine genauere Beschreibung der Parameter sei auf das separate Handbuch verwiesen. Abschließend läßt sich sagen, das beide Optimieralgorithmen einen guten Wert in kürzester Zeit gefunden haben.

10.4. Supply Chain

Bei der Optimierung des Modells der Supply Chain wurde vorrausgesetzt, daß die Kundenbestellungen ausschließlich über die Einzelhändler erfolgt. Es sollte das Lager der Einzelhändler soweit minimiert werden, das die Haltungskosten minimal werden, aber trotzdem keine Lieferzeiten für wartende Kunden anfallen. Somit wurden die abgewiesenen Anfragen von Kunden in der *FE Einzellhaendler* gezählt und anschließend geplottet.

Die größe der Lager variierte im Intervall von 40 bis 1.000. Nach einem ersten Optimierungsdurchlauf mit Hilfe des PoC/B Toolset fiel auf, daß der Bereich zwischen 40 und 400 einen sehr interessanten Verlauf aufwies (Abbildung 10.9).

Anschließend erfolgte ein zweiter Durchlauf mit einer kleineren Schrittweite im Bereich von 40 bis 400. Bei einer Lagerkapazität von 60 kristalisierte sich ein Minimum heraus

(Abbildung 10.10).

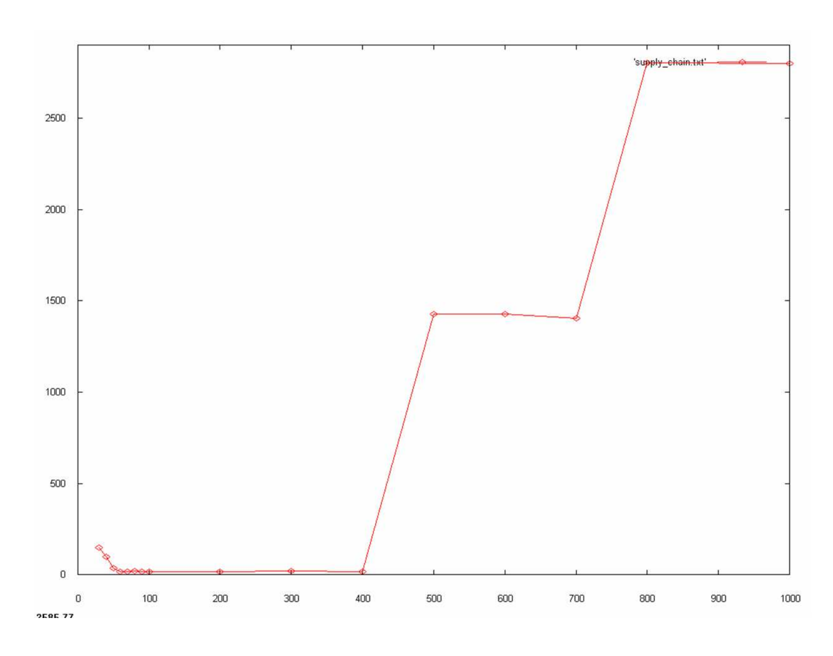


Abb. 10.9.: Response für Supply Chain

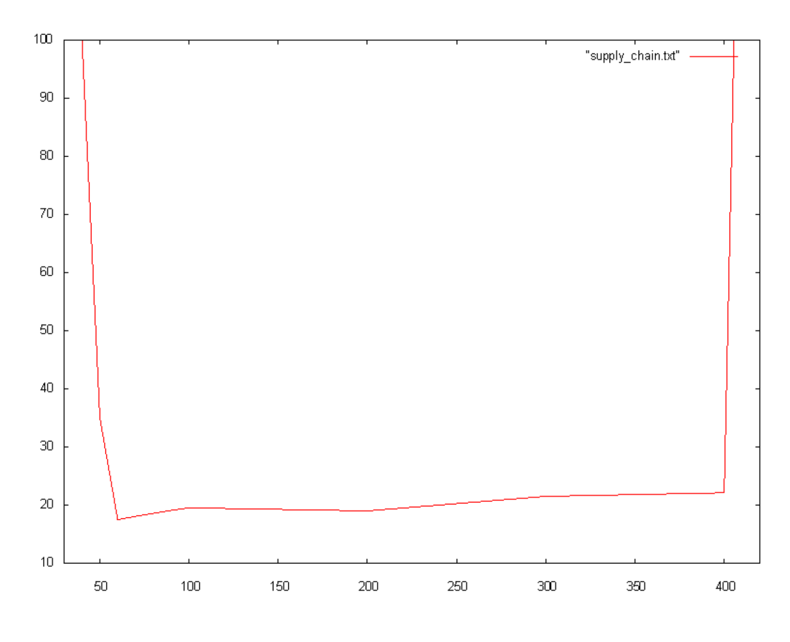


Abb. 10.10.: Response im Intervall [40;400]

Somit liegt die Optimale Puffergröße unseres Supply Chain Modells bei 60. Bei dieser Konfiguration wurden die wenigsten Kundenanfragen abgewiesen und die Haltungskosten für die Lager waren minimal.

10.5. Fertigungssystem

Beim Model des Fertigungssystems wurde versucht eine bestmögliche Konfiguration der einzelnen Stationen, bezüglich der Anzahl der Maschinen und der Größe der Puffer zu finden, damit der Gewinn maximal wird. Hier sei jedoch auf das entsprechende Kapitel verwiesen, um genauere Informationen zum Ablauf und der Responsefunktion zu erfahren. Da es sich um ein Maximierungsproblem handelt, sei kurz erwähnt, das die Responsefunktion negiert wurde, weil sowohl *RSM* als auch *EA* zur Optimierung von Minimierungsproblemen dienen. Die Anzahl der Maschinen pro Station variiert zwischen 1 bis 3. Die Puffer konnten jeweils eine Größe zwischen 1 und 10 besitzen. Somit ergibt sich ein 7-dimensionales Optimierungsproblem. Da hierfür keine Plots existieren, um Vergleiche zu anderen Konfigurationen zu schließen, wird auf die Testreihe aus dem Kapitel 9.4.4 verwiesen.

Der erste Optimierungslauf begann mit *RSM*. Ausgehend vom Startpunkt $(3, 3, 3, 3, 5, 5, 5)$ wurde mit einer Ausdehnung von $(1, 1, 1, 1, 1, 1, 1)$ versucht ein Minimum zu finden. Während der Optimierung erreichte der Algorithmus die Positionen aus 10.2

$(3, 3, 3, 3, 5, 5, 5)$
$(3, 3, 3, 3, 1, 1, 5)$
$(1, 1, 4, 2, 2, 2, 2)$

Tabelle 10.2.: Positionen des RSM Optimierers

Nach 571 Responseaufrufen verbesserte sich der Responsewert nicht mehr und der Algorithmus terminierte mit einer Konfiguration von $(1, 1, 4, 2, 2, 2, 2)$. Somit wird in Station 1 und 2 nur eine Maschine betrieben, wohingegen in Station 3 und 4 eine Maschinenbelegung von 4 und 2 optimal wäre. Die Größe der einzelnen Puffer sei überall 2. Mit der vorgegebenen Funktion wird ein Gewinn von 674.200 Euro erzielt.

Der zweite Durchlauf erfolgte mit Hilfe des *EA* Algorithmus'. Hierfür wurde die Einstellungen aus 10.3 gewählt.

BITSPERCOORDINATE	32
PPOPULATIONSIZE	10
CPOPULATIONSIZE	20
BPOPULATIONSIZE	10
NUMBEROFPARENTS	4
NUMBEROFGENERATIONS	5
THRESHOLD	0.001

Tabelle 10.3.: Konfiguration des EA Optimierers

Natürlich erfolgten noch einige weitere Einstellungen, allerdings sind hier nur einige wichtige erwähnt. Auf eine genaue Beschreibung der einzelnen Parameter sei auf das separate Handbuch verwiesen, da sie dort ausführlich erläutert werden.

Am Ende der Optimierung ergab sich eine Konfiguration für Maschinen und Puffer

von $(1, 1, 1, 2, 2, 2, 1)$. Der erwirtschaftete Gewinn beläuft sich auf 750.200 Euro. Somit erzielte der *EA* Algorithmus im Vergleich zu *RSM* eine bessere Einstellung, welche sich auch in der Testreihe im Kapitel 9.4.4 widerspiegelt. Allerdings betrug die Laufzeit der Optimierung vom Faktor Zeit her mehr als das doppelte.

10.6. Tankstelle

Beim Tankstellen-APNN-Model wurde versucht die Zwischenankunftsrate des Tankwagens und die Volumengröße des Tanks zu Optimieren. Folglich ergibt sich ein 2-dimensionales Optimierungsproblem. Die Rate des Tankwagens befand sich im Intervall von $[600; 1.500]$ Sekunden und mußte in der Transition *TwAnkommen* verändert werden. Das Volumen variierte zwischen 1.000 und 5.000 Liter. Die zuständige Transition lautet hier *AbgezogenerTreibstoff*. Um sich eine ungefähre Lage des Optimums zu verschaffen, wurde für diesen Bereich ein 3-dimensionaler Plot angefertigt, wobei auf der Z-Achse der Responsewert, also der Gewinn, aufgetragen wurde (Abbildung 10.11).

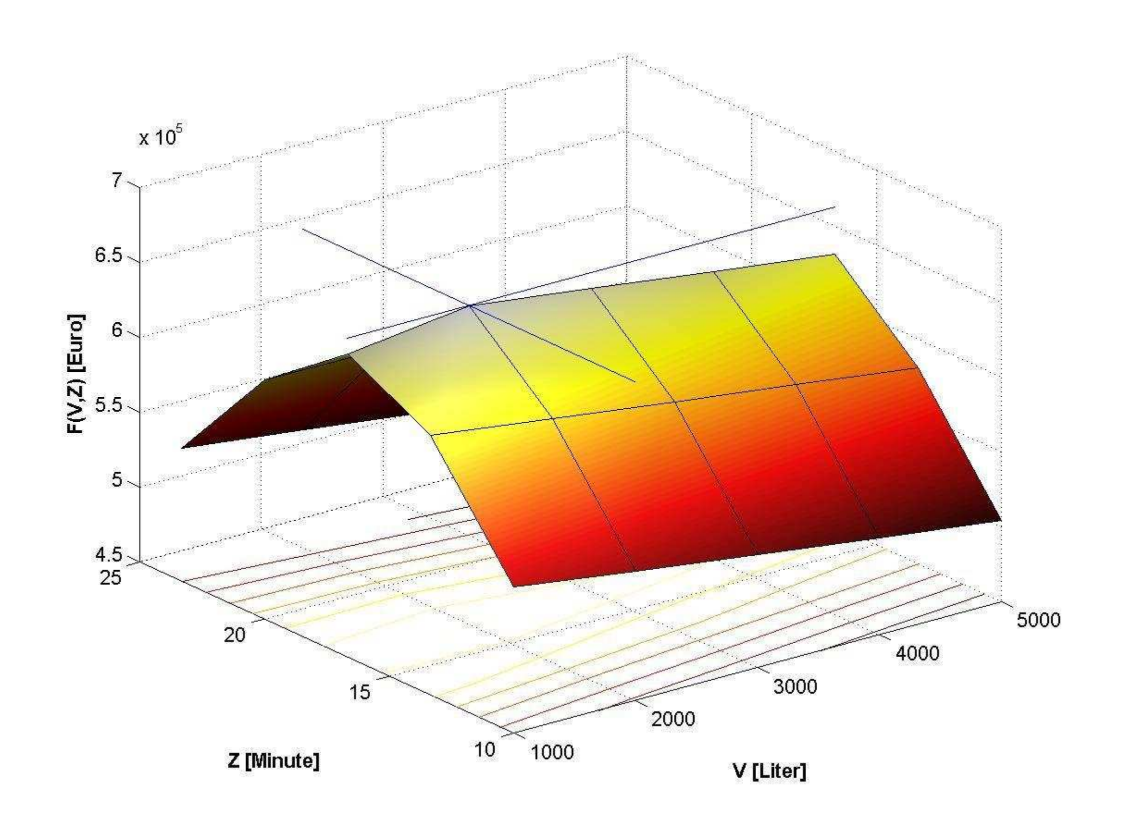


Abb. 10.11.: Responsefläche für Tankstelle

Für weitere Beschreibungen bezüglich des Models und der Responsefunktion sei auf das Kapitel 9.5.3 verwiesen.

Dieses Model wurde ausschließlich mit dem *RSM* Algorithmus optimiert. Der Startpunkt zu Beginn der Optimierung lag bei [800; 2.500] mit einer Ausdehnung von [100; 100]. Wie auch bei den anderen Modellen wurde hier ein Verkleinerungsfaktor von 20 gewählt. Der Einfluß der einzelnen Parameter auf den Optimierer sind im separaten Handbuch genauer erläutert. Nach 118 Responseaufrufen beendete sich der Algorithmus mit einem gefundenem Minimum von [1.000; 1.320]. Da der Gewinn gesucht wird und dieser normalerweise nicht minimiert werden soll, wurde ein negatives Vorzeichen vor der Responsefunktion gesetzt, da *RSM* nur in der Lage ist zu minimieren.

Das gefundene Minimum beträgt [1.000; 1.320] und ist so zu interpretieren, daß die Tankstelle bei einer Zwischenankuftszeit von 1.000 Sekunden (ca. 16,6 Minuten) und einem Tankvolumen von 1.320 Liter einen maximalen Gewinn erwirtschaftet. Vergleicht man diese Werte mit dem angefertigtem Plot, ist eine gute Übereinstimmung zu erkennen.

Teil III.

Aufbau des Programms

11. Aufbau von OaSe

Bartosz Fabianowski

Dieses Kapitel beschäftigt sich mit dem *internen Aufbau* von OaSe. Für Hinweise zur *Bedienung* siehe das separate Handbuch.

11.1. Gesamtstruktur

Bartosz Fabianowski

OaSe ist modular aufgebaut. Jedes Modul besteht aus einer oder mehreren C++ Klassen und implementiert einen logisch zusammenhängenden Teil des Gesamtsystems. Die Module kommunizieren über zuvor festgelegte enge Schnittstellen miteinander. Eine solch strikte Modularisierung hat viele Vorteile. Sie macht den Aufbau übersichtlich, sie hat uns die parallele Entwicklung der einzelnen Komponenten von OaSe ermöglicht und sie erhöht die Wartbarkeit. Abb. 11.1 gibt einen Überblick die Kommunikationsbeziehungen der Module untereinander.

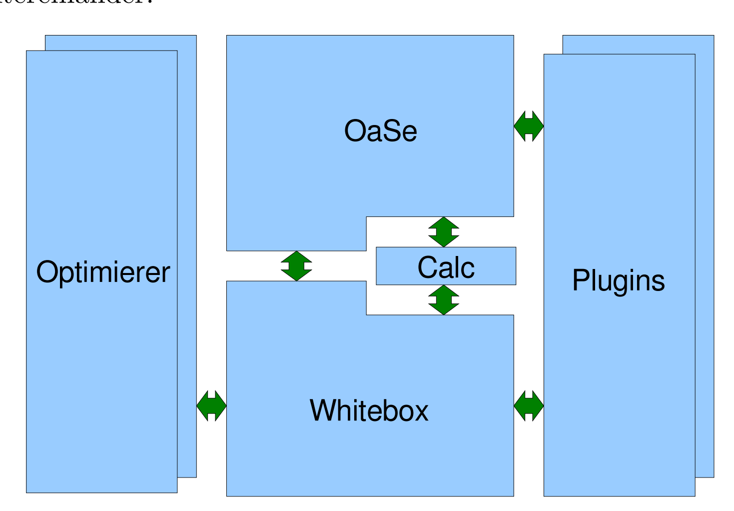


Abb. 11.1.: OaSe Modulstruktur

Die Module `OaSe`, `Calc` und `Whitebox` stellen den Kern des Systems dar. Sie bieten dem Benutzer eine GUI, nehmen seine Eingaben entgegen, validieren diese und steuern anschließend den Optimierungsprozeß. Für jede Optimierung wählt der Benutzer außerdem je ein Modul aus der Gruppe der Optimierer und der Plugins. In OaSe sind die vier Optimierer `RSM`, `EA`, `MA` und `Kriging` fest vorgegeben. Eine zukünftige Erweiterung ist wegen des großen Aufwands bei der Entwicklung eines Optimierers nicht wahrscheinlich.

Ganz anders verhält es sich mit den Plugins. Ein Plugin dient dazu, OaSe die Ansteuerung eines bestimmten Modelltyps zu ermöglichen. Sechs solche Plugins wurden von uns entwickelt und decken ein breites Spektrum ab, von einfachen Benchmarkfunktionen bis zu beliebig komplexen APNN und ProC/B Simulationsmodellen. Weil aber sehr viel mehr Modelltypen existieren und OaSe möglichst universell einsetzbar sein soll, ist die Pluginschnittstelle auf einfache Erweiterbarkeit hin ausgerichtet. So sind zur Integration eines neuen Plugins in OaSe nur zwei Codezeilen notwendig.

Im folgenden werden alle zu OaSe gehörigen Module vorgestellt und ihre Aufgaben erläutert.

11.2. Kernsystem

11.2.1. OaSe

Bartosz Fabianowski

In diesem Modul ist die für den Benutzer sichtbare GUI implementiert. Sie bedient sich der GUI Bibliothek wxWidgets, welche die Verwendung desselben Codes auf allen unterstützten Plattformen ermöglicht. Die Interaktion zwischen Benutzer und OaSe geschieht im Kontext eines Projektes. Bereits beim Anlegen des Projektes wählt der Benutzer das zu verwendende Plugin sowie ein konkretes Modell aus. Die GUI weist anschließend das Plugin an, das gewählte Modell einzulesen, zu validieren und Informationen darüber zurückzuliefern. Insbesondere sind dies die Namen der Modellvariablen und der Resultatvariablen. Die ersten beschreiben die Größen, die im Zuge der Optimierung am Modell verändert werden können. Die zweiten geben an, welche Leistungsgrößen am Ende einer Optimierung zur Verfügung stehen und zu einer Responsefunktion verknüpft werden können. Ferner kann ein Plugin dem Benutzer Optionen anbieten, eine Liste derer die GUI ebenfalls abruft und zum Aufbau der entsprechenden Bildschirmmaske verwendet. Jede Änderung einer Option durch den Benutzer wird dem Plugin kommuniziert und von diesem verifiziert. Ungültige Eingaben werden unmittelbar rückgängig gemacht.

Die eigentliche Optimierung findet auf der Basis von Faktoren statt. Diese werden vom Benutzer zunächst angelegt und mit Wertebereichen versehen. Im nächsten Schritt werden die Faktoren dann mit Modellvariablen verknüpft. Eine besondere Stärke von OaSe liegt darin, daß die Abbildung von Faktoren auf Modellvariablen nicht starr $1:1$ erfolgt, sondern extrem flexibel gestaltet werden kann. Der Wert einer Modellvariable wird im Laufe der Optimierung immer wieder aus den aktuellen Faktorenwerten berechnet. Die hierfür zu verwendende Formel legt der Benutzer fest. Bei der Festlegung der Zuordnung von Modellvariablen zu Faktoren kommuniziert die GUI daher mit dem Modul `Calc`, das jede Formel parst, auf Korrektheit überprüft und für die spätere Verwendung in geparster Form zurückliefert. Die Festlegung einer Responsefunktion auf Basis der Faktorenwerte und der Resultatvariablen geschieht analog und verwendet ebenfalls das Modul `Calc`.

Neben den Plugins bieten auch die Optimierer zahlreiche Optionen, mit deren Hilfe der Benutzer das Optimierungsverfahren an seine Wünsche und das Modell anpassen kann.

Die GUI bietet für die vier in OaSe verfügbaren Optimierer jeweils eine Bildschirmmaske mit den verfügbaren Einstellungen an. Ein großer Unterschied zu den Optionen der Plugins besteht darin, daß die hier verfügbaren Einstellungen nicht dynamisch von den Optimierern abgefragt werden, sondern im Quellcode der GUI festgelegt sind. Die Erweiterung von OaSe um einen weiteren Optimierer würde daher auch eine Erweiterung der GUI an dieser Stelle erfordern. Andererseits bedeutet dieser Aufbau, daß die GUI, wie in Abb. 11.1 dargestellt, keine direkte Schnittstelle zu den Optimierern benötigt.

Um eine Optimierung zu beginnen, übergibt die GUI die Kontrolle an die `Whitebox`. Dabei muß sie auch alle vom Benutzer vorgenommenen Einstellungen weiterreichen. Einzig die Pluginoptionen sind bereits direkt an das Plugin übergeben worden und brauchen nicht an die `Whitebox` geschickt zu werden. Die GUI teilt der `Whitebox` demnach eine Liste der Faktoren, die Formeln für die Umrechnung dieser in Modellvariablen, die Formel zur Berechnung des Responsewertes und die Einstellungen des Optimierers mit. Anschließend weist sie die GUI an, eine Optimierung mit dem vom Benutzer gewählten Optimierer durchzuführen.

11.2.2. Whitebox

Bartosz Fabianowski

Die `Whitebox` hat zwei grundlegende Funktionen. Die erste besteht darin, die zahlreichen Einstellungen des Benutzers für den Optimierer zu verwalten. Der Name `Whitebox` leitet sich von `Blackbox` ab, was den Umgang der `Whitebox` mit diesen Einstellungen am besten beschreibt. Die `Whitebox` dient als reiner Speicher, der die Daten in keiner Weise interpretiert. Sie kennt a priori weder Namen noch Datentypen der einzelnen Datensätze und erfährt diese erst, wenn die Einstellungen übergeben werden. Dieses Verhalten macht die `Whitebox` sehr flexibel und hat es uns während der Implementierung ermöglicht, den Optimierern neue Optionen zu geben ohne die Whitebox anpassen zu müssen.

Die zweite Funktion der `Whitebox` ist es, den Ablauf einer Optimierung zu steuern. Nachdem die GUI alle notwendigen Daten weitergereicht hat, übergibt die GUI die Kontrolle an die `Whitebox`. Diese wiederum teilt dem Plugin mit, daß eine Optimierung beginnt und reicht die Kontrolle anschließend sofort an den gewählten Optimierer weiter. Der Optimierer nutzt zwei Funktionen der `Whitebox`. Zunächst ruft er die Einstellungen ab, welche die GUI in der `Whitebox` für ihn hinterlegt hat. Anschließend beginnt er mit der eigentlichen Optimierung. Während dieser erzeugt er unterschiedliche Belegungen der Faktoren und benötigt für die Belegungen jeweils den Responsewert des zu optimierenden Modells. Hierfür ruft er wiederum die `Whitebox` auf.

Die Whitebox erhält vom Optimierer eine Belegung der Faktoren. Diese rechnet sie mit Hilfe des Moduls `Calc` und der vom Benutzer festgelegten Formeln in Modellvariablenwerte um. Die Modellvariablenwerte gibt sie an das Plugin weiter und veranlaßt damit eine Auswertung des Modells. Das Resultat der Auswertung ist eine Belegung der Responsevariablen. Unter Verwendung der vom Benutzer festgelegten Formel und des Moduls `Calc` bestimmt die `Whitebox` aus diesen Daten den Responsewert. Diesen gibt sie an den Optimierer zurück.

Der gesamte Prozeß ist sowohl für den Optimierer als auch für das Plugin völlig

transparent. Der Optimierer arbeitet ausschließlich mit Faktoren und einem zugehörigen Responsewert. Das Plugin dagegen verwendet Modell- und Responsevariablen. Die Vermittlung zwischen diesen beiden Sichtweisen geschieht innerhalb der `Whitebox`. Auf diese Weise brauchen weder die Entwickler des Optimierers noch die des Plugins die komplexe Funktionalität von OaSe zu berücksichtigen. Zugleich hat der Benutzer den Vorteil, daß für die Zuordnung von Faktoren zu Modellvariablen ein einheitlicher Mechanismus existiert, unabhängig vom gewählten Optimierer und Plugin.

Am Ende der Optimierung gibt der Optimierer die Kontrolle an die `Whitebox` zurück und überreicht dabei das gefundene Optimum. Die `Whitebox` wiederum reicht Kontrolle und Optimum an die GUI weiter.

Während der Optimierung baut die `Whitebox` eine Historie der bisher ausgewerteten Belegungen der Faktoren und der zugehörigen Responsewerte auf. Diese Historie wird zum einen vom `MA` Optimierer intern verwendet und kann zum anderen von der GUI abgerufen werden, um den Verlauf der Optimierung darzustellen.

11.2.3. Calc

Bartosz Fabianowski

Dieses Modul besteht aus zwei Komponenten. Der Parser prüft Formeln auf Korrektheit und überführt sie in eine maschinell schneller auswertbare geparste Form. Der Rechner kann eine zuvor geparste Formel beliebig oft mit veränderten Belegungen der enthaltenen Variablen auszuwerten.

Dem Parser wird zunächst eine Liste der gültigen Variablennamen übergeben. Anschließend können Benutzereingaben geparst werden. Erlaubt sind die vier Grundrechenarten $+$, $-$, $*$, $/$, die Klammern (und), Potenzierung ˆ, Negation durch vorangestelltes $-$, die Funktionen Absolutwert, Quadratwurzel und Logarithmus, ausgedrückt durch *%abs*, *%sqrt* und *%log*, reellwertige Konstanten und alle zuvor als gültig deklarierten Variablen. Beschreibt die Eingabe eine gültige Formel, dann wird diese in Reverse Polish Notation (RPN) ausgedrückt zurückgeliefert. Der größte Vorteil dieser Notation besteht darin, daß sie keine Klammern mehr enthält und so einfach und schnell ausgewertet werden kann. Der Parser ist mit Flex und GNU Bison realisiert.

Der Rechner benötigt eine Liste, die jeder zuvor als gültig deklarierten Variablen einen Wert zuweist. Mit dieser Liste kann er den aktuellen Wert einer geparsten Formel bestimmen.

Die Auftrennung des Moduls `Calc` in zwei Komponenten dient primär der Effizienzsteigerung. Prinzipiell wäre es selbstverständlich möglich, eine Formel immer in ihrer ursprünglichen Form zu speichern und bei jeder Auswertung neu zu parsen.

11.3. Optimierer

Bartosz Fabianowski

In diese Gruppe fallen die vier Module `RSM`, `EA`, `MA` und `Kriging`. Jedes kapselt einen der Optimierer von OaSe ein und stellt eine Schnittstelle zur Verfügung, mit deren Hilfe die `Whitebox` den Optimierer starten kann. Historisch bedingt ist diese Schnittstelle über die

vier Module hinweg nicht einheitlich. Die beiden Optimierer `RSM` und `EA` wurden im ersten Semester der Projektgruppe entwickelt. Zu diesem Zeitpunkt sollte das Modul `Blackbox` der Kommunikation mit Benutzer und Modell dienen. Als sich dieses Modul im Laufe der Entwicklung der GUI als unzureichend herausstellte, wurde mit der `Whitebox` ein leistungsfähigerer Ersatz geschaffen. Die später entwickelten Optimierer `MA` und `Kriging` kommunizieren daher direkt mit der `Whitebox`. Für die anderen beiden existiert ein Adapter, der ihnen die Schnittstelle der alten `Blackbox` auf Basis der neuen `Whitebox` zur Verfügung stellt.

Trotz technischer Unterschiede sind die Schnittstellen der vier Optimierer logisch gleich aufgebaut. Nach dem Aufruf durch die `Whitebox` entnehmen sie dieser oder der `Blackbox` die zu verwendenden Einstellungen und beginnen mit der Optimierung. Zur Auswertung des Modells rufen sie wiederum die `Whitebox` oder die `Blackbox` auf. Am Ende liefern sie das gefundene Optimum zurück. Der `MA` Optimierer, der `EA` mit `RSM` kombiniert, kann zusätzlich selbständig weitere Optimierungen anstoßen. Dazu fordert er bei der `Whitebox` mit demselben Methodenaufruf eine Optimierung an, wie die GUI ihn verwendet.

Für eine über die Festlegung der Schnittstelle hinausgehende Beschreibung der einzelnen Optimierer siehe die Kapitel 3 bis 7.

11.4. Plugins

Die sechs Plugins von OaSe gliedern sich in zwei Gruppen. Die ersten beiden Plugins ermöglichen die Verwendung von Modellen, die mit externen Werkzeugen erstellt worden sind. Für die praktische Optimierung mit OaSe sollten nur diese Plugins relevant sein. Die übrigen vier dienen dem besseren Verständnis von OaSe, der Durchführung von Tests bei Änderungen am Code und der Optimierung mit einfachen analytischen Responsefunktionen.

11.4.1. Simulationsmodelle

11.4.1.1. APNN

Bartosz Fabianowski

Dieses Plugin erlaubt es, Simulationsmodelle in Form von APNN Netzen zu laden und mit OaSe zu optimieren. Unterstützt werden beliebig komplexe zeitdiskrete Petrinetze. Die Erstellung des Modells und die eigentliche Simulation finden mit Hilfe eines externen Werkzeugs, der APNN Toolbox, statt. Das Plugin liest Modelldateien ein, schreibt sie in veränderter Form zurück, ruft den Simulator auf und interpretiert dessen Ergebnis.

Zum Einlesen von Modelldateien wird ein mit Flex und GNU Bison erstellter Parser verwendet. Dieser orientiert sich an den Parsern des APNN Editors und des Simulators. Damit ist sichergestellt, daß alle mit dem Editor erstellten Modelldateien eingelesen werden können. Auf die direkte Unterstützung hierarchischer Netze wurde zu Gunsten eines einfacheren Parsers verzichtet. Hierarchische Netze können jedoch problemlos mit

Hilfe des Editors in flacher Form exportiert werden, so daß kein Verlust an Funktionalität entsteht.

Leider existiert keine verbindliche Festlegung der APNN Grammatik. Verschiedene Werkzeuge und auch unterschiedliche Versionen desselben Programms verwenden oft kleine Abwandlungen oder Erweiterungen der ursprünglichen Grammatik. Dies betrifft auch die APNN Toolbox, deren unterschiedliche Versionen inkompatibles Verhalten zeigen. Mangels Versionsnummern ist es äußerst schwierig, die unterstützten Versionen anzugeben. Es ist daher empfehlenswert, die mit OaSe gelieferte APNN Toolbox zu verwenden. Das `APNN` Plugin ist auf Basis der darin enthaltenen Werkzeuge entwickelt und getestet worden.

Sobald eine Modelldatei eingelesen worden ist, steht das APNN Netz dem Plugin in Form einer Objekthierarchie zur Verfügung. Aus dieser werden die Modell- und Responsevariablen extrahiert. Einstellbare Größen sind die initialen Belegungen aller Stellen und die Gewichte aller Kanten. Ist das Gewicht einer Kante durch eine Zufallsverteilung gegeben, dann wird für jeden Parameter der Verteilung eine Modellvariable erzeugt. Meßbare Leistungsgrößen sind die minimale, durchschnittliche und maximale Belegung aller Stellen sowie der minimale, durchschnittliche und maximale Durchsatz aller Transitionen. Jeder extrahierten Variablen wird intern eine Referenz auf das zugehörige Objekt der Hierarchie zugeordnet.

Alle einstellbaren Parameter des APNN Simulators werden dem Benutzer in Form von Optionen angeboten. Eine weitere Option dient dazu, dem Plugin die Lage des APNN Simulator auf der lokalen Festplatte mitzuteilen.

Sobald eine Optimierung begonnen hat, führt das Plugin auf Anweisung der Whitebox Simulationen durch. Eine Simulation wird dadurch angestoßen, daß dem Plugin eine Belegung der Modellvariablen übergeben wird. Die Werte der Modellvariablen werden über die ihnen zugeordneten Referenzen in die Objekthierarchie eingetragen. Anschließend gibt die Hierarchie das durch sie beschriebene veränderte APNN Netz in eine Modelldatei aus. Mit dieser Modelldatei und den vom Benutzer vorgegebenen Einstellungen wird der APNN Simulator aufgerufen, der eine transiente Simulation durchführt.

Das Resultat der Simulation sind zwei Textdateien, die die Belegung der Stellen und den Durchsatz der Transitionen beschreiben. Das Plugin liest diese Dateien mit Hilfe eines weiteren, ebenfalls mit Flex und GNU Bison aufgebauten, Parsers ein. Die Simulation endet damit, daß die Leistungsgrößen als Belegung der Resultatvariablen an die Whitebox übergeben werden.

11.4.1.2. ProC/B

Normann Powierski

Das `ProC/B` Plugin ermöglicht die Optimierung von Modellen, die mit dem in Kapitel 8.1 vorgestellten ProC/B-Toolset erstellt wurden. Ebenso wie beim APNN-Plugin liest das ProC/B-Plugin eine zuvor ausgewählte Modelldatei und modifiziert diese anhand der übergebenen Parameter. Allerdings wird hier vor Simulationsstart erst der B1-to-Highslang-Konverter aufgerufen und erst die von diesem erstellte .hit-Datei an den Simulator übergeben. Die Ergebnisse der Simulation werden aus der vom Simulator

erstellten Ergebnisdatei ausgelesen und zur Verfügung gestellt.

Zum Parsen der ursprünglichen und Erstellen der modifizierten Modelldateien verwendet das Plugin eine leicht modifizierte Variante der Version 0.11 der B1-Schnittstelle. Dadurch ist die Auswahl verwendbarer Modelle auch auf für mit Version 0.11 erstellte beschränkt. Aufgrund des modularen Aufbaus und dem als Option einstellbaren Simulatoraufruf ist es allerdings möglich mit relativ geringem Aufwand spätere Versionen einzubinden.
Aufgrund der Komplexität der B1-Schnittstelle und der meist großen Menge von Modellvariablen wurde darauf verzichtet in Arrays oder Formeln gespeicherte Werte als optimierbare Variablen anzubieten.

Im Detail ist der Ablauf bei einer Optimierung wie folgt.
Bei der Initialisierung des Plugins wird die angegebene Modelldatei mittels der von der B1-Schnittstelle zur Verfügung gestellten Funktionalität geparst. Anschliessend werden die verschiedenen Ebenen der entstandenen Datenstruktur durchlaufen, um die Model-Variablen und -Optionen zu erhalten.
Modelloptionen (Seed, Modellzeit) werden in der GUI zusammen mit den allgemeinen Optionen angezeigt. Alle Optionen sind notwendig um den eigentlich Ablauf der Simulation zu gewährleisten. Neben den Modelloptionen zählen hierzu z.B. die Lage des Simulators und gegebenenfalls die Daten für den Remotezugang. Modelloptionen werden bei der Initialisierung mit Defaultwerten vorbelegt. Die Modellvariablen werden getrennt zur Verfügung gestellt und können vom Benutzer für die Optimierung verwendet werden. Ebenfalls aus der B1-Schnittstelle ausgelesen, werden die bereits bekannten Ergebnisvariablen, die zur Berechnung des Responsewertes über die GUI bzw. den Calculator beliebig kombiniert werden können. Nach dem Start der Optimierung erhält das Plugin von der Whitebox die Belegung dieser Modellvariablen, setzt diese über die bei der Initilisierung angelegten Referenzen zur B1 Schnittstelle und schreibt diese dann in eine neue Modelldatei für den Simulator. Ahängig vom gewählten Simulationsort (lokal oder remote) werden Konverter und Simulator gestartet.
Nach der Simulation werden die Ergenisse aus der entsprechenden Datei ausgelesen und die gewünschten Werte an die Whitebox übergeben.

Da das Proc/B Toolset inklusive dem zugehörigen Simulator nicht uneingeschränkt zugänglich ist und auch ausschließlich auf Unix/Linux-Systemen lauffähig ist, enthält das ProC/B Plugin Remotefunktionalität, die es ermöglicht die Simulation auf einem anderen Computer per SSH-Aufruf zu starten. Dazu wird vor der Simulation die Modelldatei auf den Simulationscomputer hochgeladen, dann Konverter und Simulator gestartet und schliesslich die Ergebnisdatei wieder auf den lokalen PC heruntergeladen. Zwar kostet das Remotelogin zusätzliche Zeit, allerdings ist diese meist im Verhältnis zur Simulationsdauer vernachlässigbar. Um diese zusätzliche Funktionalität auf Windows basierten Systemen nutzen zu können, werden die Dateien „plink.exe“ und „pscp.exe“ („putty“) benötigt.

11.4.2. Tests und Rapid Prototyping

11.4.2.1. Benchmarkfunktionen

Bartosz Fabianowski

Das Plugin `Benchmarkfunktionen` stellt die neun in Kapitel 2.4 beschriebenen Benchmarkfunktionen für Leistungsstudien und Tests der Optimierer zur Verfügung. Der Benutzer kann über die Optionen des Plugins die gewünschte Funktion, die Dimension sowie Art und Stärke des Rauschens festlegen. Bei Wahl der Dimension n werden zur Übergabe der Faktoren die Modellvariablen `x1` bis `x`n verwendet. Der Responsewert wird in der Resultatvariable `benchmark_response` zurückgeliefert. Die Benchmarkfunktionen sind direkt im Plugin in C++ implementiert und können deswegen sehr effizient ausgewertet werden.

11.4.2.2. Loopback

Bartosz Fabianowski

Das `Loopback` Plugin erlaubt es, ohne die Verwendung externer Modellierungswerkzeuge und ohne Programmierung beliebige analytische Responsefunktionen zu optimieren. Die Angabe einer komplexen Responsefunktion, deren Wert mit Hilfe des Moduls `Calc` aus der aktuellen Belegung der Faktoren und der Responsevariablen berechnet wird, ist bei OaSe für alle Plugins möglich. Die Aufgabe des `Loopback` Plugins besteht lediglich darin, die von `Whitebox` und GUI geforderte Minimalfunktionalität eines Plugins zur Verfügung zu stellen.

Darüber hinaus bietet `Loopback` noch einen Zufallsgenerator. Seine Saat kann über eine Option des Plugins gesetzt werden. Bei jeder Auswertung liefert er in der Responsevariable `random` eine standardnormalverteilte Zufallszahl zurück. Diese kann für die Abbildung stochastischer Einflüsse in der analytischen Responsefunktion verwendet werden.

11.4.2.3. (s,S)-Lagersystem

Bartosz Fabianowski

Dieses Plugin implementiert das in Kapitel 9.7 beschriebene (s,S)-Lagersystem. Obwohl es sich um ein Simulationsmodell handelt, benötigt eine Auswertung sehr wenig Zeit, weil der Simulator speziell auf dieses eine Modell abgestimmt und direkt im Plugin in C++ implementiert ist. Der Simulator basiert teilweise auf einer Implementierung von Axel Thümmler.

Die einstellbaren Parameter des Simulators sind über die Optionen des Plugins zugänglich. Während der Optimierung werden die veränderbaren Größen in Form der beiden Modellvariablen `s` und `S` übergeben. Das Ergebnis der Simulation steht in der Resultatvariable `costs` zur Verfügung.

11.4.2.4. Fertigungsstraße

Bartosz Fabianowski

Von diesem Plugin wird analog zum `(s,S)-Lagersystem` ein spezifisches Simulationsmodell abgebildet – die in Kapitel 9.6 beschriebene Fertigungsstraße. Auch hier kommt ein speziell auf das Modell zugeschnittener und in C++ implementierter Simulator zum Einsatz, der eine schnelle Auswertung ermöglicht. Er basiert ebenfalls teilweise auf einer Implementierung von Axel Thümmler.

Über die Pluginoptionen kann neben den Parametern des Simulators auch die Dimension n, welche die Anzahl der hintereinander geschalteten Warteschlangen angibt, festgelegt werden. Die Übergabe der Bedienraten der einzelnen Warteschlangen geschieht während der Optimierung mit Hilfe der Modellvariablen `service_rate_1` bis `service_rate_`n. Der von der Fertigungsstraße erzielte Gewinn wird in der Responsevariable `profit` zurückgeliefert.

12. Fazit

Jan Kriege

Im folgenden werden die Aktivitäten der PG 462 über die gesamte Zeit zusammenfassend dargestellt.
Es wurden vier Verfahren zur Optimierung von Simulationsmodellen implementiert: Die Response Surface Methode, das Kriging-Verfahren, Evolutionäre Algorithmen und der Memetische Algorithmus.
Die Response Surface Methode ist ein iteratives Verfahren, bei dem das Modell in jedem Schritt in einigen Punkten ausgewertet wird, um die Response Surface durch ein Polynom zu approximieren und so die Richtung des steilsten Abstiegs zu bestimmen bis ein lokales Optimum gefunden wurde.
Bei dem Kriging-Verfahren handelt es sich um ein Interpolationsverfahren, daß zwischen den Punkten eines zuvor bestimmten Experimental Designs weitere Punkte berechnet. Als Experimental Design wird Latin Hypercube Sampling verwendet und mit Hilfe des EGO-E-Algorithmus wird versucht, die Interpolation zu verbessern, indem an Punkten mit dem größten Expected Improvement ausgewertet wird.
Evolutionäre Algorithmen sind ein populationsbasiertes Verfahren, daß sich an der natürlichen Evolution orientiert und aus Initialisierung, Elternselektion, Rekombination, Mutation, Umweltselektion und verschiedenen Abbruchbedingungen besteht. Hier wurde ein Baukasten für verschiedene Suchheuristiken implementiert, der zum Beispiel die Realisierung einer typischen Evolutionsstrategie mit reellwertigen Operatoren oder eines genetischen Algorithmus mit binären Operatoren erlaubt.
Der Memetische Algorithmus realisiert eine Kombination eines Evolutionären Algorithmus mit lokalen Suchverfahren. Dazu wird jedes Individuum lokal mit RSM optimiert. Zusätzlich wurde ein adaptives Verfahren implementiert, daß Fitness-Distanz-Korrelationskoeffizienten zur Feststellung des Schwierigkeitsgrades einer Fitnessfunktion berechnet.
In die Evolutionären Algorithmen und die Response Surface Methode wurden Ranking & Selection Verfahren integriert, die aus verschiedenen vorgegebenen Konfigurationen die beste Konfiguration auswählen. Insgesamt wurden sechs Verfahren implementiert, die sich in EA in der Elternselektion, der Umweltselektion, der Verwaltung der Elitepopulation und bei der Endauswahl einer besten Konfiguration nutzen lassen. RSM nutzt die Ranking & Selection Verfahren bei der linearen Suche in Richtung des steilsten Abstiegs.

Im Rahmen von Leistungsstudien mit Benchmarkfunktionen wurden sinnvolle Parameter für alle Optimierungsverfahren ermittelt. Da die gefundenen Werte aber natürlich nicht für alle zu untersuchenden Modelle optimal sein können, hat der Nutzer die Möglich-

keit über eine GUI von den Standardwerten abweichende Konfigurationen anzugeben. Über Plugins wurden verschiedene Simulatoren an die GUI angebunden: Neben der APNN-Toolbox und dem ProC/B-Toolset, die beide am Lehrstuhl 4 entwickelt werden, und den Benchmarkfunktionen wurden noch Modelle einer Fertigungsstraße und eines (s, S)-Lagersystems integriert. Die modulare Struktur der GUI mit Plugins ermöglicht auch in Zukunft die problemlose Anbindung weiterer Simulatoren.
Mit Hilfe von APNN- und ProC/B-Modellen wurden weitere Leistungsstudien durchgeführt, um die Optimierungsverfahren anhand realer Simulationsmodelle zu testen und die Ergebnisse aus den Leistungsstudien mit Benchmarkfunktionen zu verifizieren.

Literaturverzeichnis

[1] D.V. Arnold und H.-G. Beyer, *A Comparison of Evolution Strategies with Other Direct Search Methods in the Presence of Noise*, Computational Optimization and Applications, 24. 135–159. 2003

[2] Th. Bäck, *Evolutionary Algorithms in Theory and Practice*, Oxford University Press. New York. 1996

[3] Th. Bäck und U. Hammel, *Optimierung in der Simulation: Evolutionäre Algorithmen*, Modellierung, Simulation und Künstliche Intelligenz. SCS Publishing House. Erlangen. 303–331. 2000

[4] R.R. Barton, *Designing Simulation Experiments*, Proceedings of the 2000 Winter Simulation Conference. Orlando, Florida. 45–51. 2000

[5] J. Boesel, B.L. Nelson, S.H. Kim, *Using Ranking and Selection to „Clean up“ After Simulation Optimization*, Operations Research 51. 814–825. 2003

[6] P. Buchholz, *Modellgestützte Analyse und Optimierung*, Vorlesungsunterlagen. Universität Dortmund. Dortmund. 2004

[7] P. Buchholz, A. Thümmler, *Enhancing Evolutionary Algorithms with Statistical Selection Procedures for Simulation Optimization*, Proc. Winter Simulation Conference. 2005

[8] E.J. Chen und W.D. Kelton, *An enhanced two-stage selection procedure*, Proceedings of the 2000 Winter Simulation Conference. 727–735. 2000

[9] D. Goldsman, B.L. Nelson, T. Opicka und A.A.B. Pritsker, *A ranking and selectionproject: Experiences from a university-industry collaboration*, Proceedings of the 1999 Winter Simulation Conference. 83–92. 1999

[10] Th. Jansen, *Evolutionäre Algorithmen*, Vorlesungsunterlagen. Universität Dortmund. Dortmund. 2004

[11] D.R. Jones, M. Schonlau und W.J. Welch, *Efficient Global Optimization of Expensive Black-Box Functions*, Journal of Global Optimization 13. 455–492. 1998

[12] L.W. Koenig und A. M. Law, *A procedure for selecting a subset of size m containing the l best of k independent normal populations, with applications to simulation*, Communi. Stat. Nr.14(3). 719–734. 1998

[13] A.K. Kulkarni, *ME 82 – Mechanical Engineering Measurements*, Vorlesungsunterlagen. Pennsylvania State University. University Park, Pennsylvania. 2004

[14] A.M. Law, W.D. Kelton, *Simulation Modelling and Analysis*, McGraw-Hill, 3. Auflage. 2000

[15] P. Merz, *Memetic algorithms for combinatorial optimization problems: fitness landscapes and effective search strategies*, Dissertation. Universität Siegen. 2000

[16] D.C. Montgomery, R.H. Myers, *Response Surface Methodology: Process and Product Optimization Using Designed Experiments*, 2nd Edition. John Wiley & Sons. 2002

[17] H.G. Neddermeijer und G.J. Van Oortmarssen, *A Framework for Response Surface Methodology for Simulation Optimization*, Proceedings of the 2000 Winter Simulation Conference. Orlando, Florida. 129–236. 2000

[18] B.L. Nelson, J. Swann, D. Goldsman und W.-M. Song, *Simple procedures for selecting the best system when the number of alternatives is large*, Technical Report. Department of IEMS, Northwestern University. Evanston, Illinois. 1998

[19] V. Nissen und J. Propach, *On the Robustness of Population-Based Versus Point-Based Optimization in the Presence of Noise*, IEEE Transactions on Evolutionary Computation, Nr.2. 107–119. 1998

[20] J. Pichitlamken und B.L. Nelson, *Selection-of-the-best procedures for optimization via simulation*, Proceedings of the 2001 Winter Simulation Conference. 401–407. 2001

[21] H. Pohlheim, *Evolutionäre Algorithmen*, Springer. 2000

[22] K. Weicker, *Evolutionäre Algorithmen*, Teubner. 2002

[23] R.R. Wilcox, *A table for rinott's selection procedure*, Journal of Quality Technology 16(2). 97–100. 1984

[24] NIST/SEMATECH *e-Handbook of Statistical Methods*,
http://www.itl.nist.gov/div898/handbook/,
25.07.2005

A. Benutzerhandbuch

BENUTZERHANDBUCH

Oase 1.0

Universität Dortmund
Projektgruppe 462



Systematischer Einstieg in die Optimierung mit Hilfe von Simulation
Schnellstart mit Screenshots

Marcus van Elst

Systematischer Einstieg in die Optimierung durch Simulation

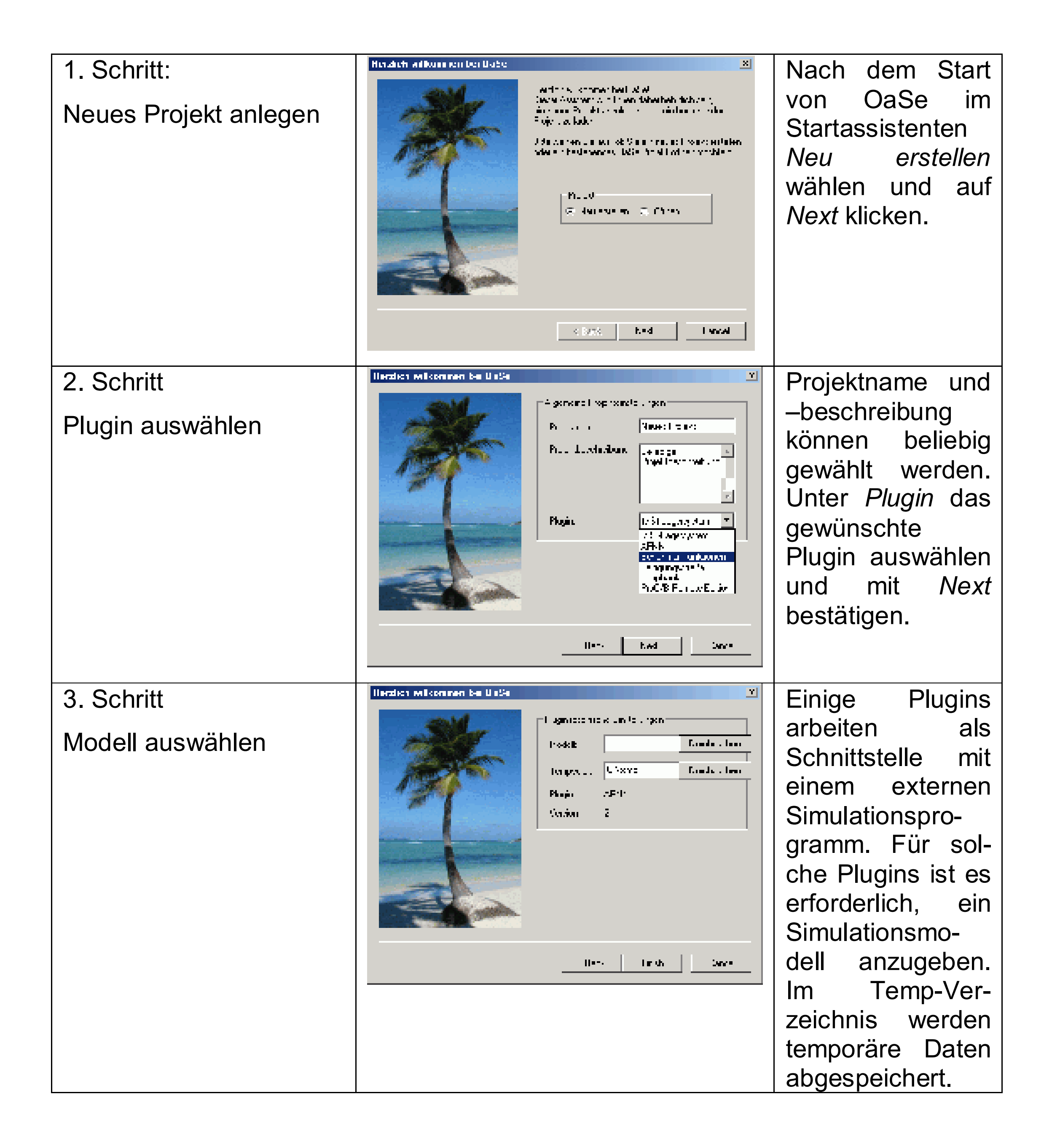
Oase 1.0

Letztes Update:
27. September 2005

Inhalt

1. Schnellstart

Dieser Schnellstart soll lediglich einen Überblick darüber verschaffen, welche Schritte notwendig sind, um eine Optimierung mittels Simulation durchzuführen. Dabei wird nur grob auf Einzelheiten, Konfigurationsparameter oder optionale Parameter eingegangen. In den folgenden Kapiteln erfolgt dann eine detaillierte Erklärung der einzelnen Konfigurationsphasen. Anhang A enthält eine Liste von Funktionen, um gültige Formeln einzugeben.

1. Schritt: Neues Projekt anlegen		Nach dem Start von OaSe im Startassistenten *Neu erstellen* wählen und auf *Next* klicken.
2. Schritt Plugin auswählen		Projektname und –beschreibung können beliebig gewählt werden. Unter *Plugin* das gewünschte Plugin auswählen und mit *Next* bestätigen.
3. Schritt Modell auswählen		Einige Plugins arbeiten als Schnittstelle mit einem externen Simulationspro-gramm. Für sol-che Plugins ist es erforderlich, ein Simulationsmo-dell anzugeben. Im Temp-Ver-zeichnis werden temporäre Daten abgespeichert.

4. Schritt Faktoren anlegen		Mindestens ein Faktor muss für einen sinnvollen Optimierungsvorgang angelegt werden. Alle erfolgreich angelegten Faktoren erscheinen nach Klicken von *Hinzufügen* in der obigen Liste.
5. Schritt Modellzuordnung vornehmen		Die Modellzuordnung stellt für manche Plugins eine optionale Einstellung dar. Durch die Modellzuordnung werden Faktoren aus Schritt 4 mit Modellvariablen verknüpft.
6. Schritt Optimierer konfigurieren		Einen der vier verfügbaren Optimierer einstellen.
7. Schritt Plugin konfigurieren		Alle Plugins verfügen über spezifische Konfigurationsparameter. Das Auswahlsymbol trägt dabei den Titel des gewählten Plugins.

8. Schritt Optimierer starten		Als letztes muss aus den angelegten Faktoren und den Modell- bzw. Pluginspezifischen Resultatvariablen eine Responsefunktion eingegeben werden, die im Folgenden minimiert wird. Hier wird auch der Optimierer ausgewählt. Nach Klicken auf *Optimieren* kann es durchaus länger dauern.

2. OaSe Projekte

Ein OaSe-Projekt besteht jeweils aus einem ausgewählten Plugin, welches als Schnittstelle zwischen Optimierer und Simulator dient und eventuell einer pluginspezifischen Modelldatei, die das eigentliche Optimierungsproblem beinhaltet. Das Plugin stellt also einen Adapter dar, mit dessen Hilfe sich praktisch beliebige Simulatoren steuern lassen können, somit kann ein Plugin als Verbindungsstück zwischen Simulator und Optimierer betrachtet werden. Das Loopback-Plugin stellt hierbei ein minimales Beispiel dar, wie eigene Plugins in C++ implementiert und eingebunden werden können. Weitere Informationen hierzu können direkt aus dem kommentierten Quellcode der Plugins entnommen werden. Eine für den jeweiligen Simulator native Modelldatei stellt ein Simulationsszenario dar, welches im weiteren Verlauf optimiert werden soll. Sobald ein Plugin im Startassistenten ausgewählt wurde, ist eine nachträgliche Änderung nicht mehr möglich.

Eine zentrale Rolle beim Erstellen eines OaSe-Projekts ist das Anlegen so genannter Faktoren. Faktoren sind die einzigen, vom Optimierer veränderbaren Parameter, die der Benutzer vor der eigentlichen Optimierung anlegt, das heißt mit einem Namen und einem Wertebereich versehen muss. Im nächsten Schritt können diese Faktoren dann mit Modellvariablen verknüpft werden, die sich wiederum aus dem ausgewählten Plugin in Kombination mit der eventuell dazugehörigen Modelldatei ergeben.

Im folgenden Schritt sollte sich der Benutzer für einen der vier vorhandenen Optimierer entscheiden und diesen konfigurieren:

- Evolutionäre Algorithmen
- Response Surface Methode
- Memetischer Algorithmus
- Kriging Metamodelle

Der letzte Schritt besteht darin, dass eine Responsefunktion aufgestellt wird. Diese wird vom jeweiligen Optimierer dann im späteren Verlauf minimiert, das heißt, es werden die Faktoren innerhalb ihres Wertebereichs verändert, so dass sich ein minimaler Responsewert ergibt.

3. Die Programmoberfläche

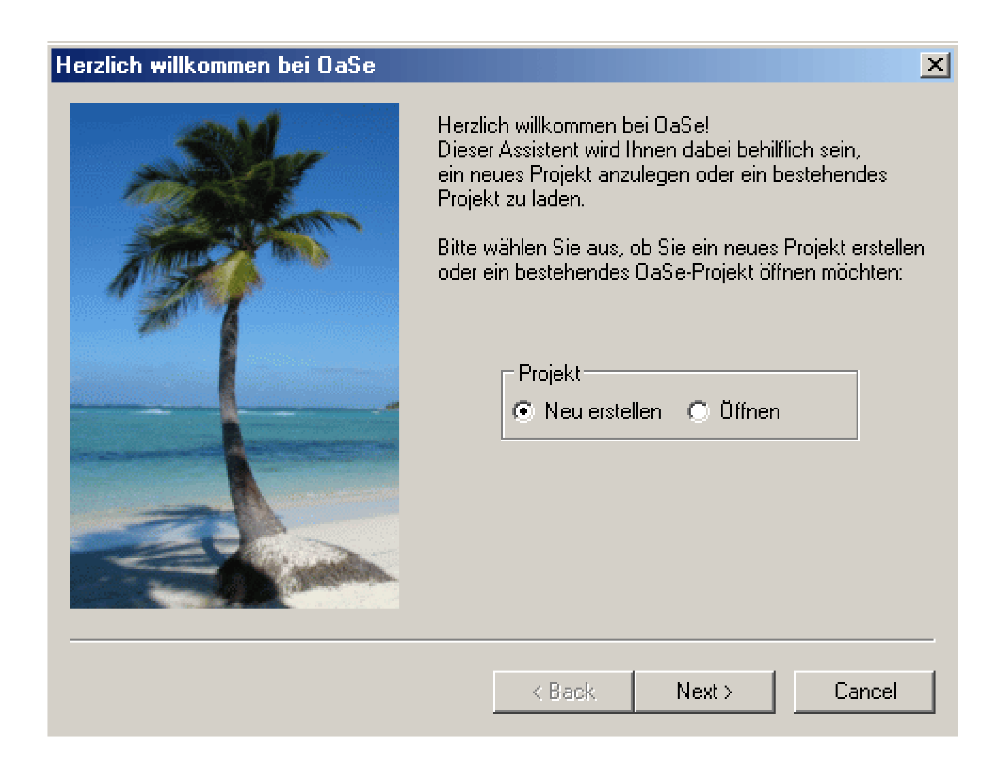
OaSe besteht grob betrachtet aus zwei Ansichten: Der Startassistent präsentiert sich bei jedem Start und ermöglicht es, ein neues Optimierungsprojekt anzulegen. Nach dem Startassistenten erscheint das Hauptfenster, welches detaillierte Einstellungen zu

- Faktoren
- Modellzuordnung der Faktoren
- Konfiguration der Optimierer
    - Evolutionäre Algorithmen
    - Response Surface Methode
    - Memetischer Algorithmus
    - Kriging Metamodelle
- Sowie Konfigurierung der derzeit vorhandenen Plugins:
    - (s,S)-Lagersystem
    - APNN
    - Benchmarkfunktionen
    - Fertigungshalle
    - Loopback
    - ProC/B Remote Edition

ermöglicht.

4. Der Startassistent

Der Startassistent dient zum Anlegen eines neuen Projekts. Die Möglichkeit, bestehende Projekte zu laden, ist zurzeit noch nicht implementiert. Als erste Auswahl bietet sich also lediglich *Neu erstellen* an:



In Schritt 2 wird nun ein Plugin ausgewählt. Ebenfalls kann das Projekt mit einem Namen und einer Beschreibung versehen werden. Die vorhandenen Plugins heißen:

- (s,S)-Lagersystem
  Dieses Plugin beschäftigt sich mit einer optimalen Lagerhaltungsstrategie. s definiert die vom Benutzer festgelegte Größe, die es zu optimieren gilt. S gibt die Grenze an, bis zu welcher Menge Waren im Lager gehalten werden sollen.
- APNN
  Dieses Plugin dient zur Optimierung von Simulationsmodellen in Form von APNN Netzen, die als native APNN Datei eingelesen werden. Unterstützt werden beliebig komplexe zeitdiskrete Petrinetze, die in einem so genannten flachen Netz vorliegen müssen. Hierarchische Netze werden nicht unterstützt, können aber von dem APNN-Editor problemlos in flache Netze umgewandelt werden.

- Benchmarkfunktionen

  Das Benchmarkfunktionen-Plugin stellt insgesamt neun komplexe Funktionen für Leistungsstudien und Tests der Optimierer zur Verfügung. Funktion sowie Dimension können unter anderem vom Benutzer ausgewählt werden.

- Fertigungshalle

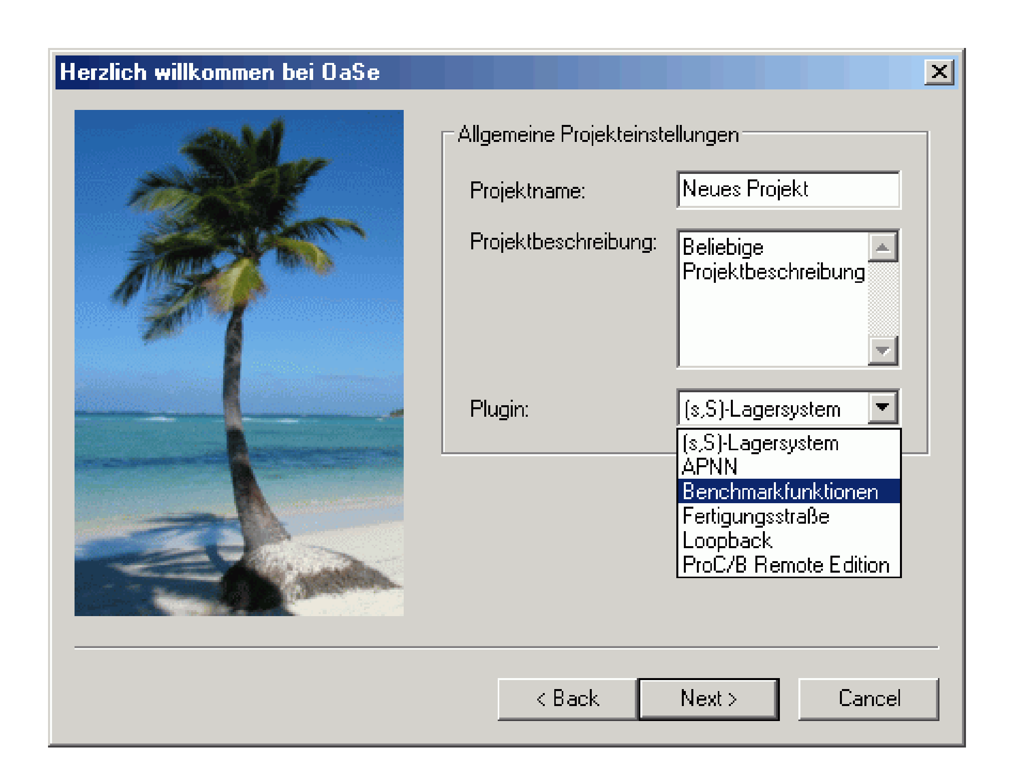
  Dieses Plugin repräsentiert ein Modell, bei dem n hintereinander geschachtelte Bedienstationen mit Kapazität 10 simuliert werden, wobei die Ankunftsrate für Kunden in die erste Bedienstation 0,5 beträgt.

- Loopback

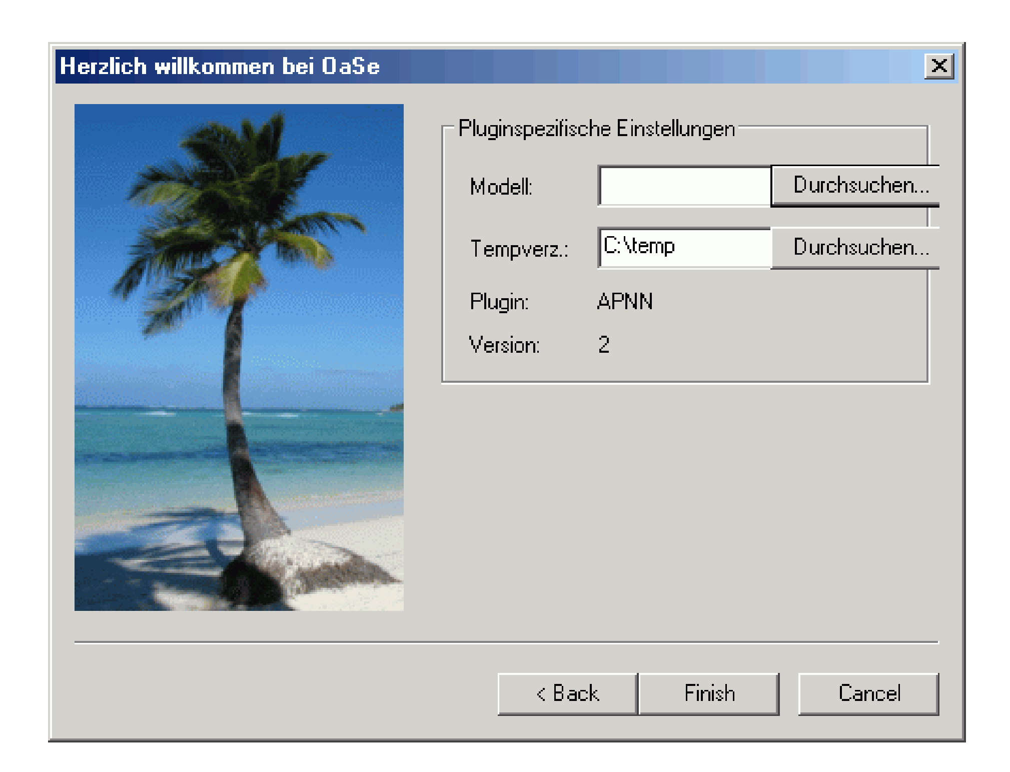
  Das Loopback Plugin dient zur Optimierung beliebiger mathematischer Funktionen, die in der Responsefunktion eingegeben werden. Die angelegten Faktoren werden eins zu eins durchgeschleift und bilden somit unmittelbar die Grundlage für die Responsefunktion, die im Folgenden optimiert wird.

- ProC/B Remote Edition

  Das ProC/B Plugin ermöglicht die Optimierung von Simulationsmodellen, die mit der ProC/B Toolbox entstanden sind. Die ProC/B Toolbox dient zum Modellieren logistischer Netze, die in der Logik als Prozessketten beschrieben werden.



Im nächsten Schritt des Startassistenten wird das ausgewählte Plugin mit der dazugehörigen Versionnummer angezeigt, ebenso wird der Benutzer aufgefordert, ein Simulationsmodell anzugeben, sofern er ProC/B Remote Edition oder APNN als Plugin ausgewählt hat. Andere Plugins benötigen keine Modelldatei, ein Temp-Verzeichnis sollte in jedem Falle angegeben werden und dient lediglich zur Zwischenspeicherung von temporären Daten.



Sobald der Benutzer nun auf *Finish* klickt, ist das Grundgerüst für das neue OaSe-Projekt fertig gestellt und Plugin- sowie Modellspezifische Parameter können konfiguriert werden.

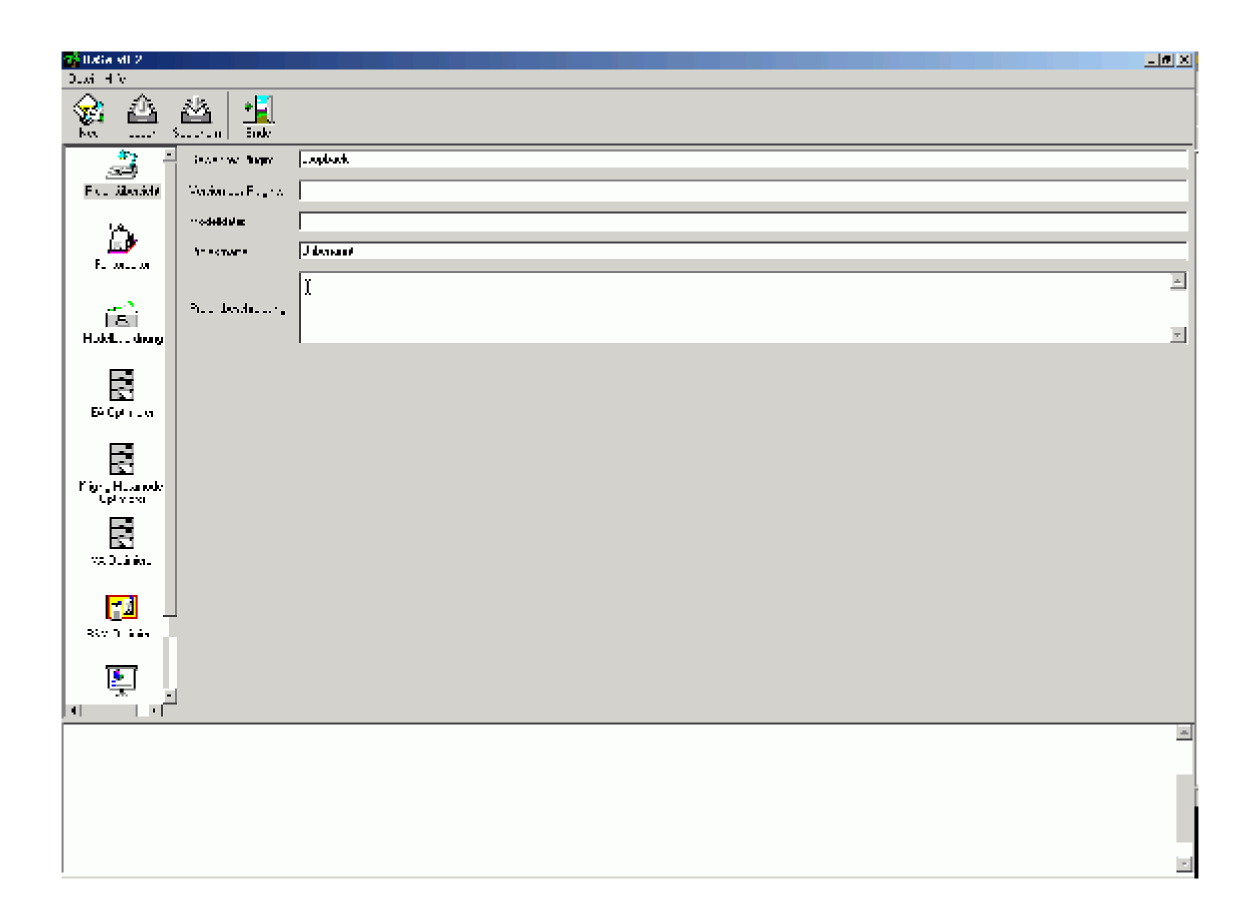
5. Das Hauptfenster

Das Hauptfenster von OaSe gliedert sich in zwei Ansichten: An der linken Seite stellen neun Icons die unterschiedlichen Konfigurationsgruppen zusammen, während im großen, rechten Bereich der eigentliche, dazugehörige Inhalt dargestellt ist. Am unteren Bildrand befindet sich zusätzlich eine Anzeige für Notizen und Fehlermeldungen. Die Icons in der Auswahlleiste haben folgende Bedeutungen:

- Projektübersicht
  Hier werden grundlegende Informationen über das Projekt angezeigt.
- Faktoreditor
  Der Faktoreditor dient zum Anlegen von Faktoren, die während der Optimierung verändert werden.
- Modellzuordnung
  Mit Hilfe der Modellzuordnung können auf flexible Art und Weise Faktoren mit Modellvariablen verknüpft werden
- EA Optimierer
  Hierunter sind geordnet nach Kategorien die Konfigurationsparameter für den EA Optimierer zusammengefasst.
- Kriging Metallmodelle Optimierer
  Hierunter sind geordnet nach Kategorien die Konfigurationsparameter für den Kriging Metallmodelle Optimierer zusammengefasst.
- MA Optimierer
  Hierunter sind geordnet nach Kategorien die Konfigurationsparameter für den EA Optimierer zusammengefasst.
- RSM Optimierer
  Hierunter sind geordnet nach Kategorien die Konfigurationsparameter für den RSM Optimierer zusammengefasst.
- Plugin
  Abhängig vom gewählten Plugin werden hier alle Einstellungsmöglichkeiten für das Plugin angezeigt.
- Optimieren
  Vor der eigentlichen Optimierung wird hier die Responsefunktion angegeben, welcher Optimierer überhaupt verwendet werden soll und ob Common Random Numbers verwendet werden.

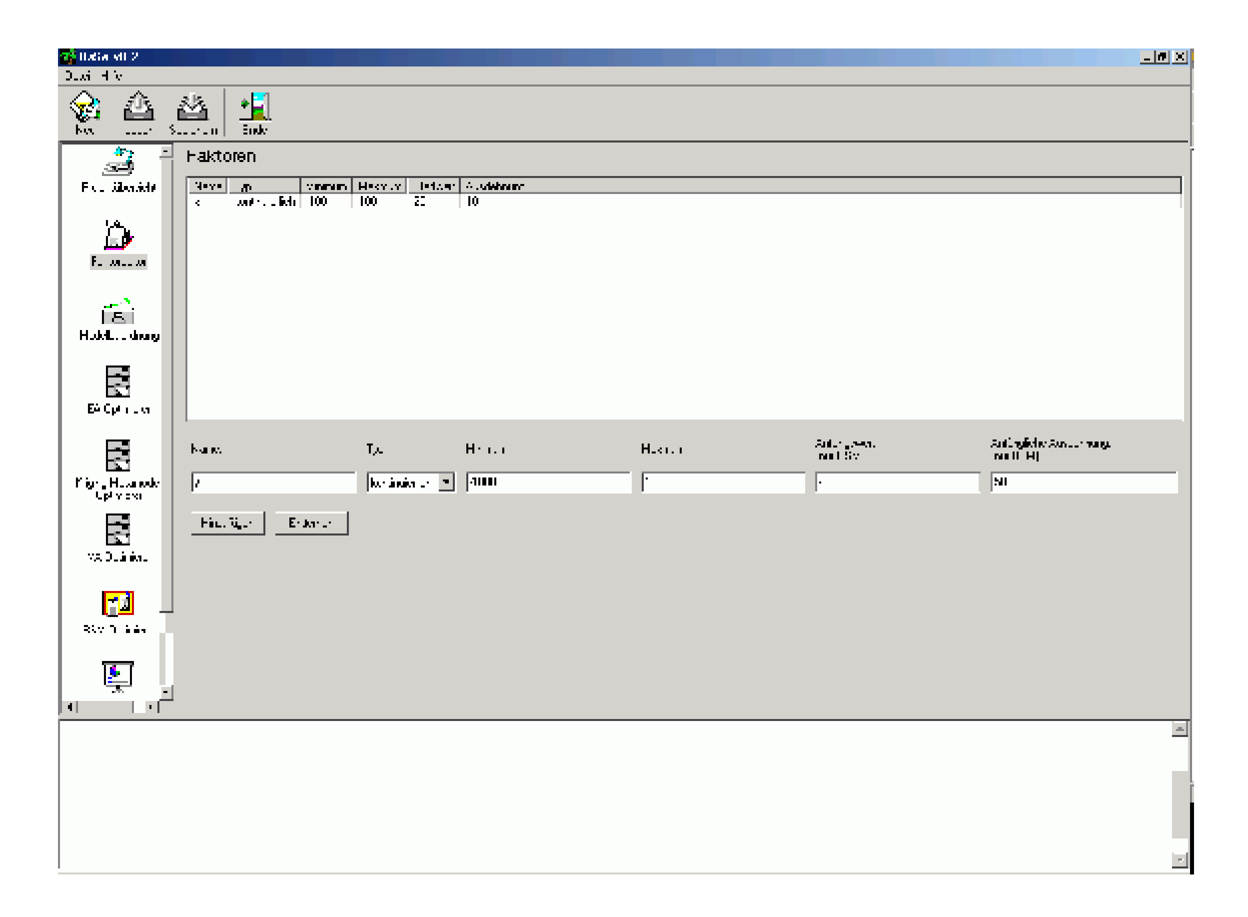
5.1. Projektübersicht

In der Projektübersicht werden Informationen über das aktuelle Projekt angezeigt. Dazu gehören: Name und Versionsnummer des ausgewählten Plugins, Pfad- und Dateiname der Modelldatei, sofern vorhanden, Projektname sowie die Projektbeschreibung. Lediglich der Projektname und die Projektbeschreibung sind in dieser Ansicht noch veränderbar.



5.2. Der Faktoreditor

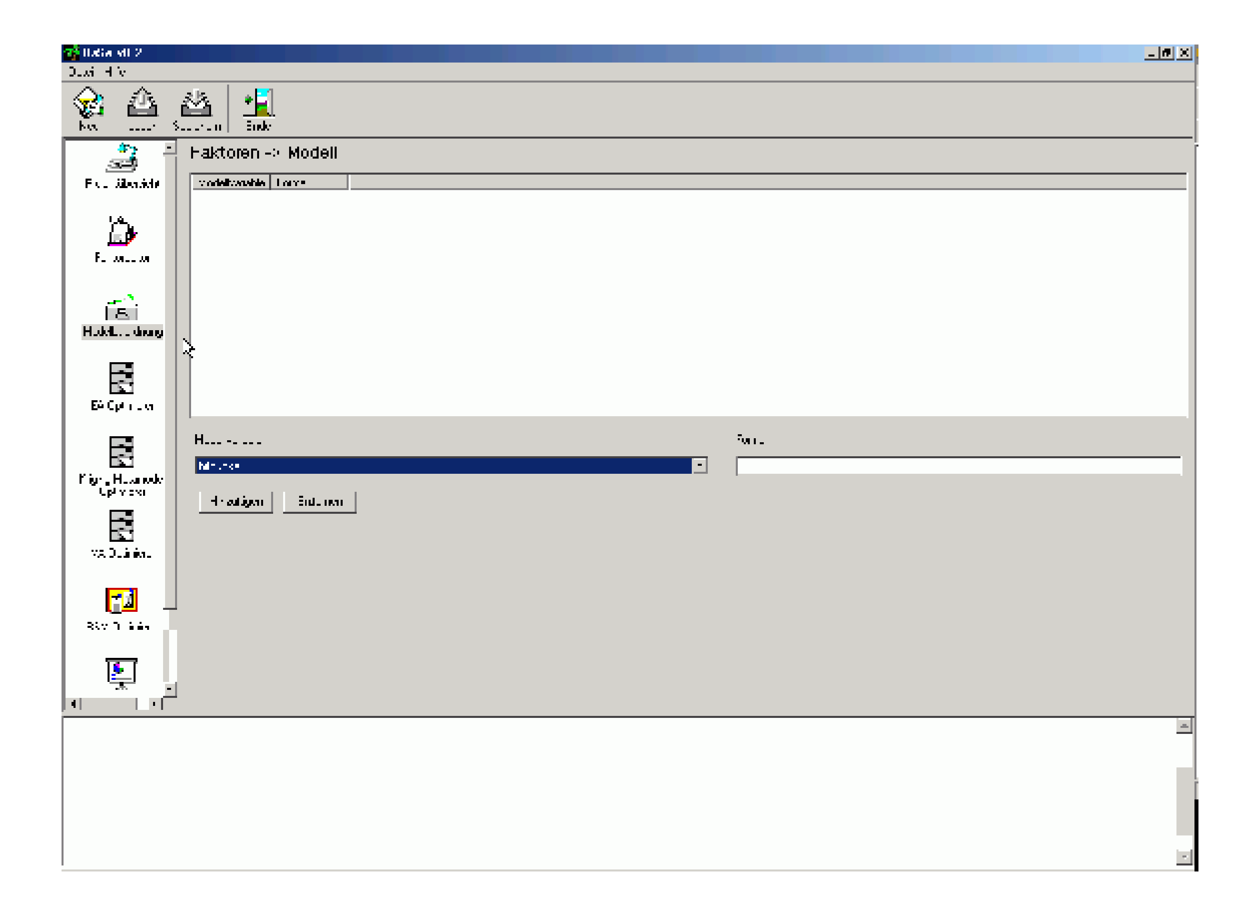
Im Faktoreditor legt der Benutzer eine Liste von Faktoren fest. Ein Faktor besteht hierbei aus einem Namen, der nicht doppelt vorkommen darf, aus der Information, ob es sich um einen diskreten, einen kontinuierlichen oder einen konstanten Faktor handelt, sowie aus einem Wertebereich, unter Berücksichtigung dessen der jeweilige Faktor veränderbar sein soll. Des Weiteren kann explizit für den RSM-Optimierer ein Startwert angegeben werden, bei dem die Optimierung beginnen wird, sowie die anfängliche Ausdehnung. Details zu den letzten beiden Parametern finden sich in dem Endbericht PG 462 in Kapitel 2.1.2. Anstelle Dezimalbrüche für Minimum, Maximum, Startwert sowie anfängliche Ausdehnung festzulegen, hat der Benutzer die Möglichkeit, eine gültige Formel wie zum Beispiel sqrt(2) = Wurzel aus 2 anzugeben. In Anhang A finden sich alle verfügbaren Funktionen und Abkürzungen, um gültige Formeln zu schreiben.



5.3. Die Modellzuordnung

Mit Hilfe der Modellzuordnung werden Faktoren mit Modellvariablen verknüpft. Die Modellvariablen ergeben sich aus dem Plugin und der verwendeten Modelldatei, die von dem jeweiligen Plugin geparst wird.

Ein großer Vorteil hierbei liegt darin, dass die Verknüpfung von Faktoren und Modellvariablen nicht eins zu eins vorgenommen werden muss, sondern diese Zuordnung mit Hilfe von Formeln vom Benutzer angegeben wird. Die Modellvariablen werden während der Optimierung also immer wieder aus den Faktoren respektive der angegebenen Formel und den Faktoren berechnet. Bei der Festlegung der Formel hat der Benutzer die Möglichkeit, auf den internen Formeleditor zurückzugreifen. Details dazu bitte Anhang A entnehmen.



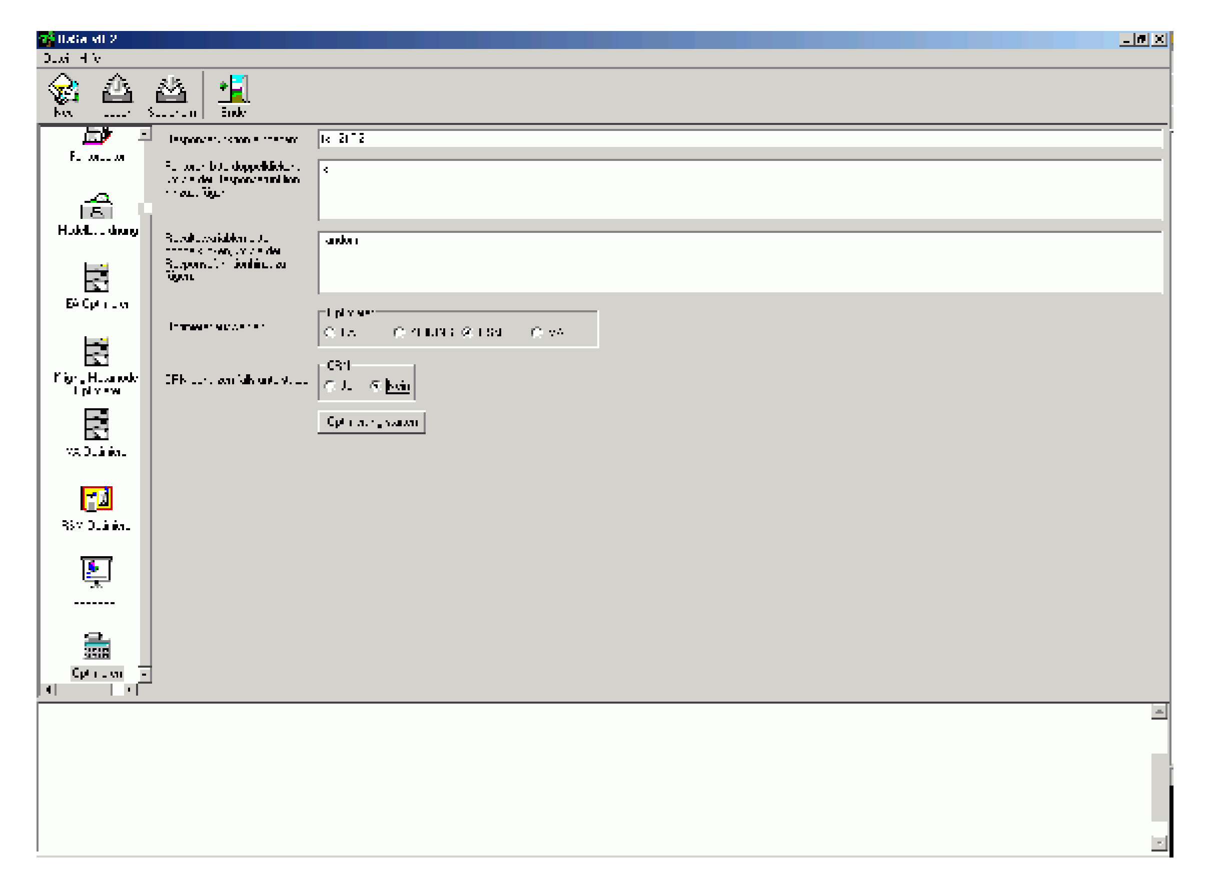
5.4. Die Optimierer

Die Ansicht Optimierer dient zum eigentlichen Starten der Optimierung, nachdem sämtliche übrigen Konfigurationsparameter, Faktoren und Modellzuordnungen gesetzt sind. Hier wird auch die Responsefunktion angegeben, deren Responsewert der ausgewählte Optimierer zu minimieren versuchen wird. Die Responsefunktion besteht dabei aus Faktoren und Resultatvariablen, die einfach über die unteren Auswahllisten per Doppelklick einzufügen sind. Dadurch erspart man sich einerseits lästige Tipparbeit und kann außerdem noch Schreibfehler vermeiden. Der Aufbau der Responsefunktion entspricht den, in Anhang A erklärten Regeln.

Nach Angabe der Responsefunktion muss man sich an dieser Stelle entscheiden, welchen der vier Optimierer zum Einsatz kommen soll. Erfahrungsgemäß liefert RSM das Optimierungsergebnis am schnellsten.

Ein letzter Parameter ist die Verwendung der Common Random Numbers (gemeinsame Zufallszahlen), die man ein- oder ausschalten kann.

Als letzten Schritt muss man lediglich noch auf *Optimieren* klicken, um die eigentliche Optimierung zu starten. Das Ergebnis kann längere Zeit dauern und wird in einer Dialogbox präsentiert.



Anhang A (Formeleditor)

Rechenzeichen

Addition	+
Subtraktion	-
Multiplikation	*
Division	/
Potenzierung	^
Klammer auf	(
Klammer zu	)

Funktionen

Absolutwert (Betrag)	%abs
Quadratwurzel	%sqrt
Logarithmus	%log

Das Argument der Funktionen wird jeweils in Klammern hinter dem Funktionsnamen übergeben. Leerzeichen zwischen den Rechenzeichen sind nicht erforderlich.